IMPROVING WEB SEARCH FOR DIFFICULT QUERIES

BY

XUANHUI WANG

B.Eng., University of Science and Technology of China, 2003
M.S., University of Illinois at Urbana-Champaign, 2006

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Doctoral Committee:

Associate Professor ChengXiang Zhai, Chair
Professor Jiawei Han
Associate Professor Kevin Chen-Chuan Chang
Research Scientist Ravi Kumar, Yahoo! Research

Abstract

Search engines have now become essential tools in all aspects of our life. Although a variety of information needs can be served very successfully, there are still a lot of queries that search engines can not answer very effectively and these queries always make users feel frustrated. Since it is quite often that users encounter such "difficult queries," improving Web search for these queries can bring significant benefits to users. However, the problem has so far been under-addressed. In this dissertation, I propose to systematically study this problem from different perspectives, naturally corresponding to different stages of an interactive search process. Specifically, I propose to improve search quality for difficult queries by: (1) Effective query reformulation, i.e., improving a search engine in the stage of query formulation. A query is difficult because it does not contain the right keywords or lacks discriminative keywords. A better formulation of query by addressing vocabulary mismatch or improving discrimination can lead to better results. (2) User-oriented search result organization, i.e., improving a search engine in result presentation. Ambiguous queries are difficult and often lead to search results with mixed senses. Search result organization can make search results easily accessible for users. (3) Incorporating user negative feedback, i.e., improving a search engine by learning from user interactions. When a query is extremely difficult and all the top results (e.g., top 10) are totally irrelevant, the feedback that a user can provide would be solely negative. I propose to develop effective negative relevance feedback strategies to improve the ranking accuracy of the next few pages when the user clicks on the "Next" button. (4) Supporting effective browsing, i.e., allowing a user to find information without queries. Browsing is a complementary mechanism to querying and is especially important for difficult queries. I propose to construct a novel multi-resolution topic map to enable effective browsing beyond hyperlinks. In summary, my dissertation is to study how to

improve search engines for difficult queries along all these four directions by exploiting both *global massive search logs* and *immediate user feedback* information.

To Qihong and Parents.

Acknowledgments

I wish to express my great thanks to all the people who gave me tremendous support and help during my Ph.D. study.

First and foremost, I am heartily thankful to my advisor, ChengXiang Zhai. This dissertation would not have been possible without his guidance. From him, I learned how to find high-impact problems, conduct rigorous research, and make effective presentations. Cheng's brilliance makes my research much more enjoyable. His persistence and dedication encourage me to go through many obstacles. For me, Cheng is not just an advisor in research. He has also broaden my knowledge and given me many advises in multiple aspects, especially on how to move beyond my comfort zone to reach my full potential. I would benefit from all these for my whole career.

I also thank my other dissertation committee members, Jiawei Han, Kevin Chang, and Ravi Kumar, for their constructive suggestions for my dissertation and invaluable help for my career. Their suggestions make my dissertation more complete and accurate. Their recommendations open up many opportunities for my career.

Furthermore, I am proud to be a member of DAIS Lab. I benefit a lot from the discussions with all of the DAISers, especially, Xifeng Yan, Dong Xin, Tao Tao, Xuehua Shen, Hui Fang, Jing Jiang, Hong Cheng, Azadeh Shakery, Bin Tan, Shui-Lung Chuang, Xu Ling, Xin He, Deng Cai, Qiaozhu Mei, Tao Cheng, Chen Chen, Tianyi Wu, Muyuan Wang, Maryam Karimzadehgan, Alexander Kotov, Yue Lu, Vinod Vydiswaran, Yuanhua Lv, and Duo Zhang.

Finally, I would like to thank Qihong for her love, faith, and confidence in me. My acknowledgements can not be finished without saying thanks to my parents for their support and encouragement. This dissertation is dedicated to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The recent decade has witnessed an explosive growth of online information, including Web pages, news articles, email messages, scientific literature, and information about all kinds of products on the Web, etc. Such large amount of data contains a lot of useful information for all kinds of human needs. Search engines are to find relevant information from large amounts of texts and thus have now become essential tools in all aspects of our life; clearly, their effectiveness would directly affect our productivity and quality of life.

The current generation search engines are very useful, but they tend to work well only for easy queries such as queries to find homepages or find popular/known topics. In general, when the user has a well-defined information need and good knowledge of target web page(s), the user can formulate an effective query and a search engine can always return relevant results on the top. In such cases, a query with a few keywords is often sufficient. Unfortunately, when a user does not have any particular target pages in mind or does not know well about the topic to be searched, as is often the case in exploratory search and informational search [41], such short keyword queries are not always effective. Technically, there might be multiple reasons why a query is ineffective [58], but three common ones are:

- **Ambiguity**. When a query contains some ambiguous words but the user is only aware of one particular sense, search results may not be optimal to the user since they are dominated by an undesired sense or mixed by multiple senses. For example, the results in the first page returned from Google for the ambiguous query "jaguar" (as of Dec. 2, 2006) contain at least four different senses of "jaguar" (i.e., car, animal, software, and a sports team); even for a more refined query such as "jaguar team picture," the results are still quite ambiguous, including at least four different jaguar

teams: a wrestling team, a jaguar car team, Southwestern College Jaguar softball team, and the Jacksonville Jaguar football team. Moreover, if one wants to find a place to download a jaguar software, a query such as "download jaguar" is also not very effective as the dominating results are about downloading jaguar brochure, jaguar wallpaper, and jaguar DVD.

- **Vocabulary mismatch**. When a user searches for very specific information in an unfamiliar domain, the user probably does not know the right terminology to describe the information need. As a result, the keywords in the query might simply do not match the terms used in the relevant documents, causing vocabulary mismatch. For example, if a user wants to find knowledge about the retrieval functions used by search engines but does not know about information retrieval terminology well, the user may use queries such as "search engine formulas," "search engine query execution methods," "search engine scoring methods," but none of them is very effective. A much more effective query would be "search engine retrieval functions."

- **Lack of discrimination**. In an unfamiliar domain, it is always hard for a user to think of additional specific terms and the resultant queries are not specific enough to pin down the relevant documents. For example, a query such as "auto quotes" can return mixed results with some about automobile insurance quotes and some about automobile sale prices. In such a case, it would be useful to add "insurance" or "sale" to make the query more discriminative and the results for a refined query will be more coherent and useful for end users.

All these difficulties may exist in exploratory or informational search when the user does not know much about a topic and has no knowledge about the existence of many distracting documents. In the end, the user often has to spend a significant amount of time in searching or simply give up. Unfortunately, such exploratory or informational queries are quite common in Web search. According to the study in [9], informational queries account for 39% to 48%. Thus, improving search accuracy for such difficult queries can potentially bring significant benefits to users.

The study of difficult queries has just started attracting attention recently, partly due to the launching of the ROBUST track in the TREC conference, which aims at studying the robustness of a retrieval model and developing effective methods for difficult queries [68, 67]. However, the most effective methods developed by the participants of the ROBUST track tend to rely on external resources (notably the Web) to perform query expansion, which has in some sense bypassed the difficulty of the problem as in reality, there is often no such external resource to exploit, or otherwise, the user would have directly gone to the external resource to find information. Indeed, the Web resource would not help improve search accuracy for difficult topics on the Web itself. Some other preliminary works on difficult queries focus on understanding why a query is difficult [27, 11, 10, 58], identifying difficult queries [67], and predicting query performance [84]. All these works are complementary to my dissertation. However, they study difficult queries in a coarse granularity: None of them has *systematically* studied difficult queries from multiple perspectives; Nor did they address the important question of how to improve Web search for difficult queries.

In this dissertation, I systematically study how to improve Web search for difficult queries from several perspectives, naturally corresponding to different stages of an interactive search process. First, the user may not be able to formulate effective queries, and thus would need support for query formulation or other ways, such as browsing, to access needed information. Second, for a difficult query, a user would have to browse through a long list of results before reaching the very first relevant page, and thus would benefit from better organization of search results. Third, it is important to learn from all the clues in a user's interaction history to better infer the user's information need, especially to perform negative feedback because of the difficulty of the query. In particular, I propose to improve a search engine through the following ways by leveraging two kinds of user information: (1) the massive amount of user queries and click-throughs; (2) the immediate feedback information from the current user:

- **Query reformulation recommendation**: Search logs contain a lot of useful knowledge such as term association patterns for query reformulation. When a difficult query is due to lack of discrimination, a user would need to add terms to make it more spe-

cific. Usually it is not easy for the user to come up with *effective* terms to add when the topic is difficult for the user. In this case, our query reformulation would recommend additional words for the user to choose, based on the term association patterns of past queries. Another factor of query difficulty is due to vocabulary mismatch and is caused by the fact that there may be multiple ways of expressing the same idea or describing the same thing (e.g., car wash, auto wash, and vehicle wash), but a user may not know what exact terms have been used by the authors of the documents to be searched. Based on the massive search logs, our idea is to identify and recommend those semantically equivalent but more popular queries to address vocabulary mismatch problem.

- **User-oriented search result organization**: When a query is ambiguous, relevant documents may be far down in the ranked list returned by a search engine. A user may need to spend a lot of time to go through a long list before reaching the very first relevant document. Search result organization is to partition the search results into coherent groups and thus can help such difficult queries. However, to be useful for end users, search result organization should be *meaningful* and *accessible* from a user's viewpoint, i.e., user-oriented. Previous methods try to cluster the search results and label each cluster solely based on their contents. Such data-oriented approaches do not consider any user preferences. In contrast, our method is to learn meaningful *subtopics/aspects* from search logs given a query topic and label each aspect by representative past queries. The obtained aspects are patterns left by past users and thus correspond to users' interested subtopics. Aspect labels are based on query words which are entered by past users and thus more accessible to end users.

- **Negative relevance feedback**: No retrieval model is able to return satisfactory results for every query. Indeed, a query might be so difficult that a large number of top-ranked documents are non-relevant. A commonly used strategy to improve search accuracy is through feedback techniques, such as relevance feedback, pseudo-relevance feedback, and implicit feedback. In the case of a difficult topic, we likely will have only negative (i.e., non-relevant) examples, raising the important question of how to

perform relevance feedback with only negative examples. We refer to this problem as *negative feedback*. Ideally, if we can perform effective negative feedback, when the user could not find any relevant document on the first page of search results, we would be able to improve the ranking of unseen results in the next few pages. I propose to conduct a systematic study of different methods for negative relevance feedback. Based on two representative retrieval models, vector space models and language models, I propose several general strategies for negative feedback and conduct systematic comparison and analysis of theses strategies on two large benchmark TREC data sets.

- **Effective browsing**: Browsing is a complementary information access mechanism to querying. When querying is ineffective, supporting effective browsing can help a user greatly. But current browsing is mainly supported through hyperlinks. I propose methods to enhance browsing by a multi-resolution topic map, which is to capture the global structure of the entire information space. At beginning, a submitted query is an anchor to bring a user to a point in the information space. When the query is difficult, it brings the user to a point that is lack of relevant information. A topic map is to help the user move from a non-promising place to a promising one. The map would allow a user to reach any interesting subtopic quickly and naturally through map operators such as "zoom in", "zoom out", and navigation into a neighbor topic. The multi-resolution makes it very easy for a user to both examine an interesting subtopic in detail (using "zoom in") and move away from a non-interesting subtopic quickly (using "zoom out" and then navigating to a different topic). At any topic, the user can open the topic to view all the documents under the topic. The map thus allows a user to tour a region of information space in the same way as a tourist uses a geographic map to tour a city. I propose to construct topic maps based on multiple resources. In the general Web search, I rely on search logs. For a document collection without search logs, I propose a keyword-oriented way to construct such maps.

As far as we know, this dissertation is the first systematic study of improving Web search for difficult queries from multiple perspectives. All the perspectives are general and the corresponding methods are novel. Since our methods are based on two types of user

logs which can be easily obtained by a search engine, they can be easily applied to any search engine. More importantly, as the search engine accumulate more logs, our methods can yield better improvement for difficult queries.

The rest of the dissertation is organized as follows. In Chapter 2, we review the related work. Section 3, 4, 5, 6, and 7 describe our approaches for query reformulation, search result organization, negative feedback, and effective browsing. We conclude this dissertation in Chapter 8.

Chapter 2

Related Work

Difficult queries remain as a big challenge for every search engine and the study of difficult queries has attracted much attention recently. In this chapter, we conduct a literature review of previous work on difficult queries.

General study of difficult queries. The launching of the ROBUST track in the TREC conference [68, 67] can be regarded as one of very first systematic effort on the study of difficult queries. The goal of ROBUST track is to study the robustness of retrieval models and developing effective methods for difficult queries. However, the most effective methods developed by the participants of the ROBUST track tend to rely on external resources (notably the Web) to perform query expansion, which has in some sense bypassed the difficulty of the problem as in reality, there is often no such external resource to exploit, or otherwise, the user would have directly gone to the external resource to find information. Indeed, the Web resource would not help improve search accuracy for difficult topics on the Web itself.

Queries are difficult due to various reasons. Understanding why a query is difficult and categorizing difficult queries into different root causes are critical to develop effective retrieval strategies. Previous work such as [27, 10, 58] identified several root causes of difficult queries based on manually examination. Savoy [58] analyzed the reasons of difficult topics from a query perspective. Harman and Buckley [27, 10] conducted the analysis from a retrieval engine perspective. However, their focus is on identifying the root causes, but do not propose methods to address those problems.

More recent work on difficult queries are on predicting query difficulty [14, 67, 84, 11]. In [14], *clarity score* is defined to measure the difference between a query language model and a collection language model. They found that the smaller the score is, the more difficult the

query is. More features are designed and analyzed in [84, 11] to estimate query difficulty. All these works predict whether a query is difficult generally but do not try to predict the root causes of why a query is difficult.

As we can see, most of previous works on difficult queries have not addressed the important question of how to improve search accuracy for difficult queries. In this dissertation, we systematically study this problem by proposing specific techniques along different perspectives. Our techniques are related to previous works as follows.

Query reformulation. Our effective query reformation techniques are related to query suggestion works such as [54] and [34]. In [54], the similarity between two queries are measured by their retrieved snippets from a search engine. In [34], adjacent query pairs from the same user sessions are used as candidates and machine learning algorithms are used to categorize query pairs into 4 classes, which reflect levels of relevance between two queries. The main difference of our work is that we primarily discover patterns in *term* level and use the discovered pattern to recommend more *effective* queries, while previous work does not consider the effectiveness of a query and only focuses on finding the generally related queries in the level of *queries*. Furthermore, they always rely on external resources such as a Web corpus or training data, while our methods only need search logs.

Our work is also related to query modification work in information retrieval community [56]. The study of query modification can be traced back to the earliest relevance feedback techniques such as the Rocchio method [53], in which queries are modified based on the documents which are judged to be relevant and irrelevant. When all the top documents are irrelevant, negative feedback can be employed [70]. Pseudo-relevance feedback is to simulate relevance feedback by assuming top ranked documents of an initial retrieval as relevant ones [81]. In [2], a system Prisma is studied and it can recommend related terms and users can narrow the search results by selecting appropriate related ones to refine their queries. All these approaches depend only on the original queries and their initial retrieved documents for query refinement. Our query reformulation algorithms are based on the term association patterns mined from many queries accumulated by search engines, thus in a *collective* and *collaborative* way. Our methods rely on users' past activities recorded

in search logs to discover term association patterns and thus can reflect users' preferences more appropriately.

Search result organization. Our work on search result organization is closely related to the study of clustering search results. In [29, 47], the authors used Scatter/Gather algorithm [17] to cluster the top documents returned from a traditional information retrieval system. Their results validate the cluster hypothesis [64] that relevant documents tend to form clusters. The system "Grouper" was described in [86, 87]. In these papers, the authors proposed to cluster the results of a real search engine based on the snippets or the contents of returned documents. Several clustering algorithms are compared and the Suffix Tree Clustering algorithm (STC) was shown to be the most effective one. They also showed that using snippets is as effective as using whole documents. However, an important challenge of document clustering is to generate meaningful labels for clusters. To overcome this difficulty, in [88], supervised learning algorithms were studied to extract meaningful phrases from the search result snippets and these phrases were then used to group search results. In [36], the authors proposed to use a *monothetic* clustering algorithm, in which a document is assigned to a cluster based on *a single* feature, to organize search results, and the single feature is used to label the corresponding cluster. Clustering search results has also attracted a lot of attention in industry and commercial Web services such as Vivisimo[1]. However, in all these works, the clusters are generated solely based on the search results. Thus the obtained clusters do not necessarily reflect users' preferences and the generated labels may not be informative from a user's viewpoint.

Methods of organizing search results based on text categorization are studied in [12, 19]. In this work, a text classifier is trained using a Web directory and search results are then classified into the predefined categories. The authors designed and studied different category interfaces and they found that category interfaces are more effective than list interfaces. However predefined categories are often too general to reflect the finer granularity aspects of a query.

Feedback techniques. Our negative feedback techniques are special cases of relevance

[1]http://vivisimo.com/

feedback. In the related literature, Feedback techniques have been extensively studied and mostly shown to be effective to improve retrieval accuracy [53, 50, 4, 55, 24, 80, 89, 61]. In general, most feedback techniques rely on positive documents – documents that are explicitly judged as relevant or implicitly assumed to be relevant – to provide useful related terms for query expansion. In contrast, negative (i.e., non-relevant) documents have not been found to be very useful. In general, exploiting non-relevant information is largely unexplored; query zone [63] appears to be the only major heuristic proposed to effectively exploit non-relevant information in document routing tasks. It showed that using non-relevant documents which are close to the original queries is more effective than using all non-relevant documents in the whole collection. However, this problem was studied for document routing tasks and a lot of relevant documents are used. Our problem setting is quite different in that we only have non-relevant documents for feedback, and we start with non-relevant documents close to a query to study how to use this negative information optimally in ad hoc retrieval. A recent work by Xu and Akella [82] studied the active relevance feedback for difficult queries. Their methods estimate both a positive model and a negative model from feedback documents and use the classic probabilistic retrieval model to rank documents. Our negative feedback techniques rely on purely negative information which is more natural due to the difficulty of queries.

Exploratory and multi-faceted search. Our work of supporting effective browsing is closely related to exploratory search which appears to have just begun to attract serious attention in related research communities [76, 77, 41, 38, 23]. Different from the previous works in HCI community which mostly focus on interface design [77, 28, 16, 25], our emphasis is to build a topic map to guide users' navigation in the whole information space. Querying and browsing are two common information search paradigms [21, 39, 40, 85]. For exploratory search, supporting browsing is important. The current researches focus on a particular domain. For example, in library science domain [6, 43, 8], a lot of catalog or meta-data information is used to support browsing. Our work is on the general Web domain and relies on unstructured search logs and support browsing for ad-hoc topics.

Hierarchies or taxonomies are always built to support browsing [35]. Traditional hier-

archies are Web directories such as Yahoo! directory and Open Directory Project (ODP). Both are built manually and only focus on *vertical* relations. As a result, current hierarchies do not have a clear notion of resolution and thus topics in the same level of a hierarchy are not always comparable with respect to granularities. Our multi-resolution topic map is an extension of traditional hierarchy but with distinct features: topics in the same level have similar granularities and horizontal links are constructed to connect related topics. Hierarchical clustering has been studied extensively in data mining communities [26]. Our methods of building multi-resolution topic maps can be a kind of hierarchical clustering, but we try to make the topics in the same level have similar granularities so that the horizontal links are more meaningful.

Faceted hierarchies [23, 83] is an extension of traditional hierarchies to support browsing. They contain multiple hierarchies along different dimensions and have been used in commercial Websites and digital libraries [60, 38]. Most of the current facets are built manually and designed specifically for a well-understood domain. Automatically constructing faceted hierarchies is admitted to be a very challenging task [23]. Some recent works such as [18] try to automatically extract facet terms in a text database with certain progress. However, it is still unclear about how to build faceted hierarchies for an ad-hoc topic. With the same goal of supporting browsing, our topic maps extend traditional hierarchies in a way orthogonal to faceted hierarchies. Different from faceted hierarchies, our topic maps can be applied to a general domain to support ad-hoc queries.

While hierarchy and faceted hierarchies are generally used to support a user to exploit *inside* the search results. Our topic maps can enable a user to *horizontally* navigate to neighbor areas which is *outside* of the current search results.

Other Related Work. Our methods are based on search logs. In the past, search logs have been exploited for several different purposes, such as clustering search queries to find those Frequent Asked Questions (FAQ) [74, 7]. Recently, search logs have been used for suggesting query substitutes [34, 54], personalized search [61], Web site design [5], Latent Semantic Analysis [71], and learning retrieval ranking functions [48, 32, 1].

The notion of information footprints has been used in some previous work such as [75],

where footprints were also used to build maps, trials, and annotations to help a new user for information exploration. Our novelty lies in that we treat search logs as footprints and propose algorithms to turn search logs into a multi-resolution topic map to support flexible browsing in the entire information space. Some other works such as [45, 44] do not create new ways to support browsing, but try to make the existing hyperlinks easier for users to browse. Our work on topic maps is to break the limitation of hyperlink following through more flexible browsing with a topic map.

Chapter 3

Effective Query Reformulation

3.1 Introduction

The first stage in a search cycle is for a user to formulate a query. Mis-specification and under-specification (corresponding to vocabulary mismatch and lack of discrimination) are two common problems in query formulation when a user searches in an unfamiliar domain. In this chapter, we propose to mine massive search logs to discover *term* level association patterns to address these two problems so as to help users in this initial stage of the search cycle.

As search engines are being used, they naturally accumulate a lot of log data, including submitted queries, viewed search results, and clicked URLs. Such search engine logs contain a lot of valuable information such as patterns of query reformulation. In general, a Web search engine answers millions of queries every day. Thus the huge amount of search engine log data offers excellent opportunities for data mining. Indeed, mining search engine logs has recently attracted much attention [61, 48, 32, 1, 15, 72, 59]. All these studies have shown the promise of improving search accuracy through mining search engine logs. However, virtually all the previous work has treated *a whole query* as a unit for analysis; as a result, the discovered knowledge is mostly at the level of queries. For example, clustering search queries is studied in [74, 7]. The similarity of queries can be measured by the clicked documents [74] or their temporal correlations [13, 66]. Existing query suggestion works such as [54] and [34] also consider a whole query as a unit and they further rely on other resources such as Web snippets [54] or human-labeled training data [34] to generate related queries. Furthermore, most of the work only suggests "related" queries and does not consider the effectiveness of the suggested queries, which is very crucial for successful query suggestions.

In this chapter, we look into patterns at the level of *terms* through analyzing the relations of terms *inside* a query and use the discovered term association patterns for *effective* query reformulation. Our work is motivated from the following observations about what types of knowledge are useful to help a user formulate an effective query. A query is ineffective due to multiple reasons, but two of them are common: *mis-specification* and *under-specification*.

(1) The mis-specification problem is caused by the fact that there may be multiple ways of expressing the same idea or describing the same thing, and a user may not know what exact terms have been used by the authors of the documents to be searched. This is also called "vocabulary mismatch." For example, if a user wants to find a place to wash his/her vehicle, a good query would be "car wash". If the user uses a query such as "auto wash" or "vehicle wash", the search results are generally not as good as those from using the query "car wash" even though all these queries have roughly the same meaning. This is because in most relevant web pages, the authors used "car wash" rather than "vehicle wash" or "auto wash." In order to help a user in such a case, we need knowledge of the form "auto→car | _ wash" (i.e., in the context "_ wash", it is better to replace "auto" with "car"). This is an example of what we refer to as a context-sensitive term substitution pattern.

(2) The under-specification problem in a query may be because the user does not know much about the content to be found or can not naturally think of additional specific terms. For example, a query such as "auto quotes" can return mixed results with some about automobile insurance quotes and some about automobile sale prices. In such a case, it would be useful to suggest terms such as "insurance" and "sale" for a user to choose so as to make the query more discriminative. In order to do this, we need knowledge of the form "+insurance | auto _ quotes" and "+sale | auto _ quotes" (i.e., in the context of "auto _ quotes", "insurance" and "sale" are possibly useful terms to refine the query at a specified position). This is an example of what we refer to as a context-sensitive term addition pattern.

In this chapter, we first formally define the two novel term association patterns in search logs – context-sensitive term substitution and addition patterns. Then we propose new probabilistic methods to discover these patterns through analyzing term co-occurrences in query

logs. Our basic idea is to analyze the co-occurrences of terms within multi-word queries in logs and obtain two kinds of term relations: (1) quasi-synonyms and (2) contextual terms. Quasi-synonyms are words that are synonyms (e.g., auto and car) or that are syntactically substitutable (e.g., yahoo and google) [30]. Such terms tend to co-occur with the same or similar terms; for example, both "auto" and "car" often occur together with "rental", "pricing", etc. We propose to use probabilistic translation models for capturing quasi-synonyms. Contextual terms are terms that appear together. For example, "car" and "insurance" often co-occur in the queries and they can help each other to refine a topic – "car insurance" can be used to refine both "car" and "insurance". We propose to use probabilistic contextual models for capturing contextual terms. Based on both translation models and contextual models, we cast our context-sensitive term association pattern mining as probability estimation problems. Patterns with high probabilities are with high confidence and then used for query reformulation. For example, "car" has a high probability in the translation model of "auto" and high probability to co-occur with "wash" in contextual models, then the pattern "auto→car|_ wash" will have a high probability and thus is a pattern with high confidence.

To test the effectiveness of our proposed algorithms, we conduct experiments on a sample of search logs. Experimental results on the real search engine logs show that our proposed methods can efficiently and effectively mine term association patterns and all these patterns can be used for effective query reformulation. These show that our proposed methods can discover useful knowledge based on the term relations inside queries. Our methods are totally orthogonal to, and thus can be enhanced by, other techniques which use other information such as click-through and user session data for query suggestions.

The rest of the chapter is organized as follows. We formally define our mining problem in Section 3.2 and propose our models to discover term association patterns in Section 3.3. Our search log data collection is described in Section 3.4 and the experiments are presented in Section 3.5. Finally we conclude this chapter and discuss future work in Section 3.6.

Queries	Clicked URLs	Time
hotel taxes in las vegas	http://xxx.xxx.xxx/	xxxx
las airport	/* no clicks */	xxxx
las vegas airport	http://xxx.xxx.xxx/	xxxx
	http://xxx.xxx.xxx/	xxxx
...	...	...

Table 3.1: An dummy example of user sessions

3.2 Problem Formulation

Search engine logs record the activities of web users, which reflect the actual general users' need or interests when conducting a search. Generally, search engine logs have the following information: text queries that users submitted, the time when they searched, and the URLs that they clicked after the queries. Search engine logs are separated by user sessions. A user session includes several queries from the same user for a coherent information need and the clicked URLs for each query in the session. An example of user sessions is shown in Table 3.1. In this chapter, we focus on the pattern inside *queries* in search logs and we formally define our problem of mining term association patterns in this section.

Definition 1 (Query) *A query q of length n with vocabulary V is an ordered sequence of terms $[w_1 w_2 ... w_n]$, where $w_i \in V$ for all $1 \leq i \leq n$. We use $w \in q$ if term w is contained in q.*

Definition 2 (Query Collection) *A query collection Q consists of a bag of N queries: $Q = \{q_1, q_2, ..., q_N\}$. The queries are not necessarily distinct from each other.*

For example, all the queries submitted to a search engine in a certain period of time form a query collection. A query collection provides us data for mining term association patterns. We now define two interesting patterns in search logs.

Definition 3 (Context-Sensitive Term Substitution) *A context-sensitive term substitution pattern is in the form of $[w \rightarrow w' | c_L \_ c_R]$. c_L and c_R are left and right context words and this pattern means that term w should be substituted by term w' given a specific context.*

Definition 4 (Context-Sensitive Term Addition) *A context-sensitive term addition pattern is in the form of* $[+w|c_L\_c_R]$. *This pattern means that term* w *can be added into the context* $c_L\_c_R$ *and thus forms a new sequence* $c_L w c_R$.

The defined term association patterns can be easily extended for query reformulation. We define two types of query reformulation, query rewording and query refinement, in the following and they are to address the mis-specification and under-specification problems of an ineffective query respectively.

Definition 5 (Query Rewording) *Given a query* $q = w_1 w_2 ... w_n$, *query rewording is to modify the query by replacing one term* w_i *in* q *by its semantically similar term* s, *thus form a new query* $q' = w_1 ... w_{i-1}, s, w_{i+1} ... w_n$.

Definition 6 (Query Refinement) *Given a query* $q = w_1 w_2 ... w_n$, *query refinement is to modify the query by adding one semantically related term* r *to* q *before a position* i, *thus form a new query* $q' = w_1 ... w_{i-1} r w_i ... w_n$.

It can be seen that query rewording and query refinement correspond to context-sensitive term substitution and term addition patterns respectively. In practice, query rewording and refinement involve multiple terms. We only consider single terms in the consideration of complexity.

3.3 Term Association Pattern Mining from Search Logs

In this section, we first define two basic types of relationship between a pair of terms: syntagmatic and paradigmatic relation [49], which correspond to our contextual and translation models respectively. We then describe our term association mining approaches based on these two models. In the following, we use $c(x, X)$ to represent the count of x in collection X.

3.3.1 Contextual and Translation Models

Contextual Models

Our contextual models are designed to capture syntagmatic relations between terms. The syntagmatic relation is for those terms which frequently co-occur together. For example "rental" has a stronger syntagmatic relation with "car" than the word "basketball" since "rental" co-occurs with "car" more frequently in queries. In general, semantically related terms have stronger syntagmatic relation. This type of knowledge is very useful for query refinement. We first define term contexts.

Definition 7 (Term Contexts) *Given a query collection Q and a term w, we have several different types of contexts for w.*

General Context G is a bag of words that co-occur with w in Q. That is, $a \in G \Leftrightarrow \exists q \in Q$, s.t. $a \in q$ and $w \in q$.

The i-th Left Context L_i is a bag of words that occur at the i-th position away from w on its left side in any $q \in Q$.

The i-th Right Context R_i is a bag of words that occur at the i-th position away from w on its right side in any $q \in Q$.

For example, given that a query "national car rental" appears in the query collection, "national" and "rental" are in the general context G of "car"; only "national" is in the L_1 and only "rental" is in the R_1 of "car". L_i and R_i are more precise contexts for each term. In the following, given a type of context C, we use $C(w)$ to represents w's C context.

Our contextual models are to capture the syntagmatic relations probabilistically. Given a term w, different terms have different strength of syntagmatic relation with w. We thus model this relation probabilistically and adopt language model approaches here: Given a word w and its context $C(w)$, the contextual model is a uni-gram language model. The Maximum Likelihood estimation (ML) is

$$P_C(a|w) = \frac{c(a, C(w))}{\sum_i c(i, C(w))}.$$

Intuitively, a context model tells us what words have high probabilities to appear around a given word w (or at a specific position).

Smoothing techniques are usually used for language models due to the data sparseness problem. An effective approach is Dirichlet prior smoothing [90]:

$$\tilde{P}_C(a|w) = \frac{c(a, C(w)) + \mu P(a|\theta_B)}{\sum_i c(i, C(w)) + \mu}$$

where $P(a|\theta_B)$ is a predefined reference model (usually set as the whole collection language model) and μ is the Dirichlet prior parameter to be set empirically (3000 in our experiments). Note that we use $\tilde{P}_C(\cdot|w)$ and $P_C(\cdot|w)$ to represent the smoothed and non-smoothed contextual models of w respectively.

Translation Models

Our translation models are designed to capture paradigmatic relations between terms. The paradigmatic relations capture words which are quasi-synonyms (e.g., "car" and "auto"). Our translation models are built on contextual models. The basic idea is that two terms have stronger paradigmatic relation if they have similar contexts. For example, "car" and "auto" may share a lot of contextual words such as "sales" and "insurance" and thus have strong paradigmatic relation. This type of knowledge could be very useful in helping a user replace a query term (with a potentially better term) in a certain context.

In our translation model, we use $t(s|w)$ to denote the probability of "translating" w to the word s. In the language modeling approach, we use the Kullback-Leibler divergence (KL) $D(\cdot||\cdot)$ between two contextual models to measure the similarity between two contexts. Given two language models p and q, their KL-divergence is defined as

$$D(p||q) = \sum_u p(u) \log \frac{p(u)}{q(u)}.$$

The KL-divergence value is smaller if p and q are similar. We use KL-divergence on con-

textual models to define $t_C(s|w)$ as follows:

$$t_C(s|w) = \frac{\exp(-D[P_C(\cdot|w)||\tilde{P}_C(\cdot|s)])}{\sum_s \exp(-D[P_C(\cdot|w)||\tilde{P}_C(\cdot|s)])}.$$

After a few transformations, it can be seen that

$$t_C(s|w) \propto \prod_u \tilde{P}_C(u|s)^{c(u,C(w))}$$

which is the likelihood of generating s's context $C(s)$ from w's smoothed contextual model. The above formula can be applied on any type of contextual models. For example, we can use contexts G, L_1, or R_1. In this chapter, we use a combination of L_1 and R_1 contexts since these two are most indicative of the word in consideration:

$$t(s|w) = \frac{|L_1(w)| \times t_{L_1}(s|w) + |R_1(w)| \times t_{R_1}(s|w)}{|L_1(w)| + |R_1(w)|} \tag{3.1}$$

where $|L_1(w)|$ ($|R_1(w)|$) is the total number of terms occurring in the L_1 (R_1) context of w.

3.3.2 Mining Term Substitution Patterns

The context-sensitive term substitution patterns give us knowledge about query rewording. Recall that query rewording is to substitute a term w_i in q to be s and thus we get another query $q' = w_1...w_{i-1}sw_{i+1}...w_n$. The new query q' should have similar/related meaning to q. A good substitution should require that q' is better than q to retrieve more relevant documents. In other words, s is more appropriate than w_i to capture the information need given the context words in the query. For example, "car wash" is generally better than "auto wash" in the context "_wash".

Basic Approaches

In order to discover term substitution patterns, the probability we are interested in is: substituting the i-th position word w_i by s given the context $w_1...w_{i-1}\_w_{i+1}...w_n$. We use

a shorthand $t(w_i \rightarrow s|q)$ to represent this probability. Then

$$\begin{aligned} t(w_i \rightarrow s|q) &= P(s|w_i; w_1...w_{i-1}\_w_{i+1}...w_n) \\ &\propto P(w_i; w_1...w_{i-1}\_w_{i+1}...w_n|s)P(s) \\ &\propto t(w_i|s)P(s)P(w_1...w_{i-1}\_w_{i+1}...w_n|s) \\ &= t(s|w_i)P(w_1...w_{i-1}\_w_{i+1}...w_n|s) \end{aligned} \tag{3.2}$$

In Equation 3.2, each substitution candidate s is scored based on two factors: The first factor $t(s|w_i)$ reflects the similarity between the word w_i and the candidate s. This is a *global* factor which tells us how globally similar these two words are. The second factor $P(w_1...w_{i-1}\_w_{i+1}...w_n|s)$ is the *local* factor based on other context words in the query q. It tells us how likely s appears in such a context defined by q. These two factors are combined together to score each candidate in our method. To estimate the second factor, a simple approach is assume the context words are independent from each other given s. Using the general context G, we have

$$P(w_1...w_{i-1}\_w_{i+1}...w_n|s) = \prod_{j=1,j\neq i}^{n} \tilde{P}_G(w_j|s)$$

The general context ignores the position information and also considers all the words in context. In general, a word far away for the position in consideration should have lower impact. We thus use the more precise contextual models L_i and R_i and ignore those words which are far away:

$$\prod_{j=1,i-j>0}^{k} \tilde{P}_{L_{i-j}}(w_{i-j}|s) \times \prod_{j=1,i+j\leq n}^{k} \tilde{P}_{L_{i+j}}(w_{i+j}|s) \tag{3.3}$$

where k is the number of adjacent terms to consider. For example, if we set $k = 2$, we have $P(w_1...w_{i-1}\_w_{i+1}...w_n|s)$ as

$$\tilde{P}_{L_2}(w_{i-2}|s)\tilde{P}_{L_1}(w_{i-1}|s)\tilde{P}_{R_1}(w_{i+1}|s)\tilde{P}_{R_2}(w_{i+2}|s). \tag{3.4}$$

Note that we always use smoothed contextual models in the above formulas. To make the

distributions at different position i comparable, we use n-th root of the value in Equation 3.3 and n is the total number of factors in the product.

Estimation Enhanced by User Sessions

For term substitutions, we need a reliable translation model $t(s|w)$. However, estimating $t(s|w)$ based only on contextual models need to be limited to ensure that s and w are semantically similar. For example, "American idol" and "American express" are two popular queries. Thus the same word "American" can show up in the L_1 contexts of "idol" and "express" frequently; as a result, we will have a high translation probability of t(express|idol), which is not desirable. To improve the translation models, we rely on the user sessions in our search logs (see an example in Table 3.1). Since queries in a user session are usually coherent, we would expect "idol" and "express" would not appear in the same sessions very often.

We use Mutual Information (MI) of the two words s and t over sessions to measure their correlation. MI is widely used to measure the mutual independency of two random variables in information theory, which intuitively measures how much information a random variable tells about the other. In our case, MI can be computed as follows:

$$I(s,w) = \sum_{X_s, X_w \in \{0,1\}} P(X_s, X_w) \log \frac{P(X_s, X_w)}{P(X_s)P(X_u)}. \tag{3.5}$$

where X_s and X_w are two binary random variables corresponding to the presence/absence of term s and term w in each user session. For example, $P(X_s = 1, X_w = 1)$ can be calculated as the proportion of the user sessions in which s and w are both present.

To make MI comparable across different pairs of words, we use a normalized version of MI in our chapter, which is defined as

$$NMI(s,w) = \frac{I(s,w)}{I(w,w)} \tag{3.6}$$

It is easy to verify that $NMI(w,w) = 1$ and $0 \leq NMI(s,w) \leq 1$.

For our term substitution pattern mining, we combine $t(s|w_i)$ and $NMI(s,w_i)$ as follows:

(1) Given q and w_i, we use $t(s|w_i)$ to find the top N words which have the highest probabilities in $t(s|w_i)$.

(2) For each of these N words, we calculate its NMI with w_i using session information.

(3) We use a threshold τ to remove a word s from the N words if $NMI(s, w_i) \leq \tau$.

In our experiments, we set $N = 20$ and $\tau = 0.001$. Since all the remaining words have high translation probabilities and frequently co-occur with w in user sessions, they are more reliable and we thus set all $t(s|w_i) = 1$ for those remaining terms and compute $t(w_i \rightarrow s|q)$ only based on Equation 3.3. To decide when we need to replace w_i by s, we use $\frac{t(w_i \rightarrow s|q)}{t(w_i \rightarrow w_i|q)}$ as an indicator. For example, if this value is larger than 1, we would recommend to replace w_i by s.

A query may contain multiple words and any one of them can be potentially replaced/reworded. In an interactive manner, a user can tell the system which term he/she wants to replace. In an automatic manner, our general strategy is to iterate all the words and try to replace each of them. Then we get a set of candidates which differ from the original query by one term. Each of these candidates has a probability computed by Equation 3.3. We finally sort all these candidates by their corresponding probabilities and recommend the top ranked ones as substitutions.

3.3.3 Mining Term Addition Patterns

A term addition pattern $[+w|c_L\_c_R]$ is to add a word w given the context $c_L\_c_R$. Formally, given a query $q = w_1w_2...w_n$ which contains n words and a position i, our task is to recommend a term r which can be added to the original query to form a new query $q' = w_1...w_{i-1}rw_i...w_n$. We formalize this problem in a probabilistic way and we use $\gamma_i(r|q)$ to denote the probability of a pattern $[+r|w_1...w_{i-1}\_w_i...w_n]$.

$$\begin{aligned} \gamma_i(r|q) &= P(r|w_1...w_{i-1}\_w_i...w_n) \\ &\propto P(w_1...w_{i-1}\_w_i...w_n|r)P(r) \end{aligned}$$

$P(w_1...w_{i-1}\_w_i...w_n|r)$ can be estimated similarly to the estimation of the local factor in Equation 3.2. Here we estimate it similarly to Equation 3.3 as follows:

$$\prod_{j=1,i-j>0}^{k} \tilde{P}_{L_{i-j}}(w_{i-j}|r) \cdot \prod_{j=0,i+j\leq n}^{k-1} \tilde{P}_{L_{i+j}}(w_{i+j}|r) \tag{3.7}$$

$P(r)$ is the prior probability of the appearance of the term r. In the simplest case, we can assume $P(r)$ to be uniform thus it will not affect the ranking of different terms.

Intuitively, the terms which have higher probabilities to be added to q are those which co-occur frequently in the query collection together with the words in q. Each word will be assigned a probability based on Equation 3.7 and we then rank all the terms.

Given query $q = w_1...w_n$, there are $n + 1$ positions in which we can add a term. In our experiments, we iterate over all these positions and get a list of new query candidates for position i. Each query has a corresponding probability estimated from $\gamma_i(r|q)$. A final list of the recommended queries is the ranked list merged from queries for all the positions.

3.4 Data Collection

We construct our data set based on the MSN search log data set released by the Microsoft Live Labs in 2006 [42]. Our log data spans 31 days from 05/01/2006 to 05/31/2006. In total, there are 8,144K queries, 3,441K distinct queries, 4,649K distinct URLs, and 7,470K user sessions in the raw data.

We separate the whole data set into two parts according to the time: the first 2/3 data is used to simulate the history data and it is used as a query collection to mine the term association patterns. The queries in the last 1/3 data are retained to test our methods. In the history collection, we clean the data by only keeping those well-formatted English queries (queries only containing characters from 'a' to 'z' and space). We also use a predefined stopword list to remove those common words such as "a" and "the" from our query collection. After cleaning, we get 4,431,152 queries in our query collection in total and 1,577,424 of them are distinct. The total number of unique words contained by the queries in this collection is 199,629 and the media length of the queries is 2. This data set

is used in our experiments to compute the contextual models and translation models. For the user sessions, we obtain 3,540K in total and 1,320K of them have at least two queries in our training data. We use these 1,320K user sessions to compute the normalized mutual information between two terms.

Based the queries in our query collection, we build G, L_2, L_1, R_1, and R_2 contexts for the 76,693 most frequent words in the collection. All the contexts provide us the statistics of the necessary probabilities needed in our contextual models. Furthermore, we also compute the words which have high translation probabilities to each of the 76,693 words and thus build their translation models. All the contextual models and translation models are stored for online query rewording and query refinement.

3.5 Experiments

In this section, we describe our experiments on mining term association patterns from the search engine logs. In all the following experiments, we set smoothing parameter $\mu = 3000$ and $k = 2$ in Equation 3.3.

3.5.1 Contextual and Translation Models

In Table 3.2, we show the G, L_1 and R_1 contextual models of two words: "car" and "yahoo". From this table, we can see that all the contextual models in this table appear to be meaningful. We can also see that G contexts mix L_1 and R_1 contexts and that L_1 and R_1 contexts are much different. This shows it is better to model these precise contexts for a given term in our query collection. Furthermore, these contextual words may cover different aspects. In Table 3.3, We show the results of the discovered aspects of "car" and "yahoo". The aspects are obtained by applying the star clustering [3] on the top words in the G contextual models (see [3] for more details) and we show the top 5 clusters. Clearly, for the word "car", people usually care about "rental" and "pricing". In the example of "yahoo", people are interested in "search", or "games". All these aspects correspond to different information needs of end users and thus can be potentially used for search result organization and query refinement.

w=car					
a	$P_G(a\|w)$	a	$P_{L_1}(a\|w)$	a	$P_{R_1}(a\|w)$
rental	0.152	rent	0.1187	rental	0.1937
rent	0.046	rental	0.0892	rentals	0.0517
rentals	0.0318	national	0.0656	seat	0.044
enterprise	0.0306	classic	0.0451	audio	0.0403
national	0.0301	enterprise	0.0396	dealers	0.0312
prices	0.0271	race	0.0257	insurance	0.031
audio	0.0246	budget	0.0237	wash	0.0275
budget	0.0197	alamo	0.0235	max	0.0251
insurance	0.0192	electric	0.0182	sales	0.0246
dealers	0.0191	hertz	0.0169	loan	0.0203
w=yahoo					
a	$P_G(a\|w)$	a	$P_{L_1}(a\|w)$	a	$P_{R_1}(a\|w)$
mail	0.4806	sbc	0.5853	mail	0.5316
games	0.0672	verizon	0.0366	games	0.073
maps	0.0459	mail	0.0327	maps	0.0506
finance	0.0402	launch	0.0274	finance	0.0444
music	0.0354	sign	0.0228	music	0.0384
sbc	0.0331	email	0.015	personals	0.0259
personals	0.0234	chat	0.0124	email	0.0213
email	0.0206	download	0.0124	messenger	0.0157
messenger	0.0157	games	0.0111	sports	0.0138
sports	0.0136	weather	0.0111	chat	0.0133

Table 3.2: Examples of the contextual models for "car" and "yahoo".

w=car		w=yahoo	
1.	buy, prices, values	1.	search, people, address
2.	rental, rent, alamo	2.	news, sports, photos
3.	audio, stereo, speakers	3.	online, games, word
4.	accidents, crashes	4.	videos, music, video
5.	loans, calculator, payment	5.	messenger, instant, im

Table 3.3: Aspects for words "car" and "yahoo".

w=fox		w=bmw		w=computer	
s	$t(s\|w)$	s	$t(s\|w)$	s	$t(s\|w)$
fox	0.0024	bmw	0.00195	computer	0.00155
cbs	0.00035	honda	0.00019	computers	0.00014
cnn	0.00034	suzuki	0.00017	laptop	0.00011
abc	0.00032	yamaha	0.00017	pc	0.00009
bbc	0.0003	triumph	0.00017	notebook	0.00009
w=leg		w=chinese		w=calcium	
s	$t(s\|w)$	s	$t(s\|w)$	s	$t(s\|w)$
leg	0.00571	chinese	0.0027	calcium	0.01034
abdominal	0.00024	japanese	0.00011	sodium	0.00037
stomach	0.00024	korean	0.0001	potassium	0.00035
legs	0.00024	italian	0.00009	magnesium	0.0003
muscle	0.00019	greek	0.00009	cholesterol	0.00023

Table 3.4: Examples of translation models of 6 different terms.

We now show several examples of our translation models using Equation 3.1. In Table 3.4, we give 6 different terms from different domains and their translation models. For each term example, the top 5 words with highest translation probabilities are shown. We can seen that our proposed translation models are very effective to identify semantically similar words. For example, "fox", "abc", and "cnn" are all related to broadcast companies; "bmw", "honda", and "suzuki" are motorcycle/car brands. It is also interesting to note that words about different languages and words about different minerals can be identified to be similar. All these show the effectiveness of our proposed translation models to identify 'related words". All these terms provide possibility for users to do exploratory search.

3.5.2 Term Substitution Patterns

In this section, we study the effectiveness of our term substitution patterns. We first show several examples. We then compare our methods with previous methods to show that our method can improve the effectiveness of a query.

In this section, we enhance our translation models using user session information. Table 3.5 show an example. It can be seen that the words reranked using Normalized Mutual Information can indeed reduce those non-related words.

Translation Model		Mutual Information	
s	$t(s\|w)$	s	$NMI(s,w)$
idol	0.0149626	idol	1
express	0.00305314	idols	0.00270233
airlines	0.00207636	top	0.000339295
inventor	0.00195964	medical	0.000206774
haunting	0.00194115	west	1.70E-04

Table 3.5: Translation model and Normalized Mutual Information of w = "idol".

Pattern	**Reworded query**
auto→car |_ wash	car wash
car→auto |_ trade	auto trade
children→kids |_ games	kids games
kids→children |_ clothing	children clothing
driving→maps | google_	google maps
military→army |_ acu	army acu
birthday→greeting |_ cards	greeting cards
lotto→lottery | florida _ results	florida lottery results
interpretation→meanings | _ of dreams	meanings of dreams
music→song |_ lyrics	song lyrics

Table 3.6: Examples of term substitution patterns.

Examples of Substitution Patterns

We show several substitution patterns on query rewording. We decide to reword a query if the ratio $\frac{t(w \to s|q)}{t(w \to w|q)} > 1$, which means that s is more likely than w given query q. Table 3.6 shows several examples of the patterns (1st column) and the reworded queries by our method (2nd column). From the table, we have the following observations: (1) Our method can recommend more effective queries. For example, "kids games" is usually more effective than "children games". (2) Term substitution patterns are context-sensitive. For example, we substitute "auto" by "car" in the context "_ wash", while we substitute "car" by "auto" in the context of "_ trade". (3) We can see that queries from our methods are related to the original ones, but their meanings are not exactly equivalent (e.g., "birthday cards" and "greeting cards"). This is because translation models tend to find words with similar concepts but not always having the exactly same meanings, in general.

Effectiveness Comparison I

In this section, we study the effectiveness of query rewording by comparing with a previous method proposed in [34].

Experiment Design. In [34], related queries are generated according to user sessions. For each query, they first find all its next queries in all user sessions and use Log-Likelihood Ratio (LLR) to identify those highly related queries. Then they rerank the queries based on a model learned from training data. In our work, we use their linear regression model for reranking and use LLR to denote this method. The LLR method gives a ranked list of queries. Some of the resulting queries is query rewording, but they also contain other types of query reformulation such as query refinement. To compare fairly, we filter the ranked list and only retain those queries which are rewording of the original queries.

To compare different methods, we construct our test cases from our hold-out logs as follows:

1) We merged all the sessions which have the same initial queries together as one test case.

2) For each test case, we use all the clicked URLs, *except* those of the initial query, in all the merged sessions to approximate relevance documents. These documents are to approximate relevant documents which users obtained through query reformulation. Our purpose is to compare different methods with respect to fetching additional relevant documents.

Given the test cases constructed above, we compare 3 methods: The first method (denoted by Original query) is to use the initial/original queries to get a ranked list from a search engine. The second method is the LLR method in [34] and the third is ours. For either of these two methods, we first generate a list of recommended queries. We then fetch a ranked list of search results for each of the queries from the same search engine. Finally we use our relevance judgement to evaluate these different search results.

Our goal is to test which method can recommend more effective queries. Since both our and LLR methods can not generate recommended queries for every query, we thus filter the test cases and only retain those for which both LLR and our method can generate at least 5 queries and for which the first generated queries by our method satisfy $\frac{t(w \to s|q)}{t(w \to w|q)} > 1$. This

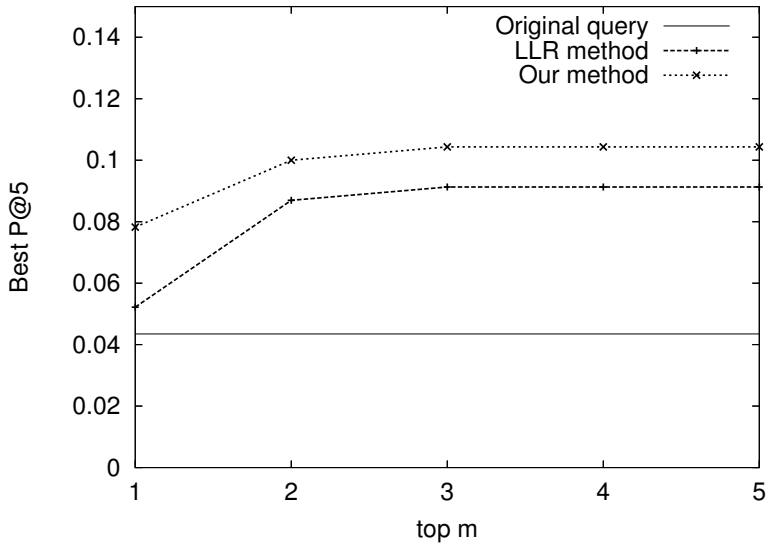

Figure 3.1: Comparison of term substitution patterns. We compare the best P@5 of the top m recommended queries by different methods.

is to simulate the scenario in which the first query is not very effective since it is precisely in such a scenario that a user would need help with reformulation. From all the remaining test cases after filtering, we randomly sample 50 test cases for our evaluation.

We use Precision@5 (P@5) as our evaluation metric. Given m recommended queries for each test case, we select the best one which has the largest number of relevant documents in its top 100 search results and use its P@5 as the accuracy of the corresponding test case.

Results. We vary the number of recommended queries m from 1 to 5 and the best P@5 values are shown in Figure 3.1. From this figure, we can see that both our method and LLR outperform original query and thus can recommend more meaningful queries. Compared with LLR method, our method is more effective. For example, when we only consider the first recommended query, our method can give P@5 = 0.08 while LLR method can only give P@5 = 0.05. This is because LLR method does not consider the effectiveness of a query while our method would recommend a more effective one based on the term substitution patterns mined from search logs. For example, our method can recommend "cheap tickets" for "cheap airfare", while LLR only suggests queries such as "discount airfare", which is not as effective as "cheap tickets".

translation pairs					
plural/singular		**abbreviation**		**others**	
map	maps	dept	department	hair	hairstyles
page	pages	tx	texas	space	myspace
code	codes	tv	television	pics	pictures
number	numbers	co	company	fish	fishing
loan	loans	st	saint	air	airlines

Table 3.7: The categories and examples of translation pair after applying lexical matching constraint.

Effectiveness Comparison II

In this section, we study the effectiveness of our method for queries with the same meanings. In order to ensure that the recommended queries have the same meanings as the original ones, we propose a lexical matching constraint for the translation pairs.

Lexical Matching Constraint. Given a word w, we first get its top 3 words with the highest probabilities based on its translation model. This gives us 3 translation pairs. Our lexical matching constraint only retains those pairs such that every character in the short word of the pair must appear in the long word, in the same order. For example, "tx" and "texas" are a qualified pair since "t" and "x" both appear in "texas" and "t" is before "x" in both words. By applying this lexical matching constraint, we can hopefully get semantically equivalent pairs.

Table 3.7 shows several examples of the identified pairs after applying our lexical matching constraint. We found that the results can be classified into three categories: plural/singular, abbreviations, and others.

Using the translation pairs filtered by our lexical matching constraint as candidate pairs, we apply our query rewording algorithm on a set of queries sampled from our test data. Each of these queries q contains at least a word w from our candidate pairs and our rewording algorithm tries to replace w with its paired word s. If the ratio $\frac{t(w \to s|q)}{t(w \to w|q)} > 1$, we reword q by replacing w by s. Finally, we rewrite 1,437 queries.

Table 3.8 shows several examples of our query rewording, ordered by their ratios. In this table, we can see that some queries are changed from singular to plural form, while some queries are changed from plural to singular form. All the changes are context sensitive. For

Original query	New query	Ratio
maps quest	map quest	358.571
sams clubs	sams club	264.500
white page	white pages	149.027
six flag	six flags	39.2353
aol email	aol mail	31.7024
continental air	continental airlines	21.2667
hair pics	hair pictures	20.0000
lotto tx	lotto texas	16.4815
window media	windows media	14.4026
yahoo map	yahoo maps	7.96296

Table 3.8: Examples of context-sensitive query rewording using the pairs filtered by our lexical matching constraint.

example, our algorithm changes "yahoo map" to "yahoo maps" (from singular to plural), but changes "maps quest" to "map quest" (from plural to singular). Intuitively, our reworded queries are more effective since the domain names of these two queries are maps.yahoo.com and mapquest.com. A recent work [46] has reach similar conclusion that context-sensitive stemming can improve click-through rate. Our results are consistent with this conclusion.

Effectiveness Experiment Design. To study the effectiveness of query rewording, we use the clicked web pages in our search engine log data to evaluate. Given a query, we collect all the positions of its clicked documents in our test log data and aggregate all these clicks together. We treat all the clicked positions as the positions of relevant documents in a ranking list and evaluate and compare the accuracy of the original queries and our recommended queries. Since we use our lexical constraint to force all the pairs to share equivalent meaning, comparing their results to show their effectiveness is reasonable. Intuitively, a better query would retrieve more relevant documents on the top of the ranked list that users tend to click. In our experiments, for each query, we calculate the precision at 1, 5, 10, 15, and 20 documents.

Effectiveness Results. Figure 3.2 shows the comparison between the original queries and the reworded queries. The precisions are averaged over all the queries that our algorithm decides to rewrite. Clearly, we can see that, at every level of precision, our recommended queries can retrieve more relevant documents and thus outperform the original queries. For example, the P@10 of our recommended queries is 0.42, while the P@10 of the original

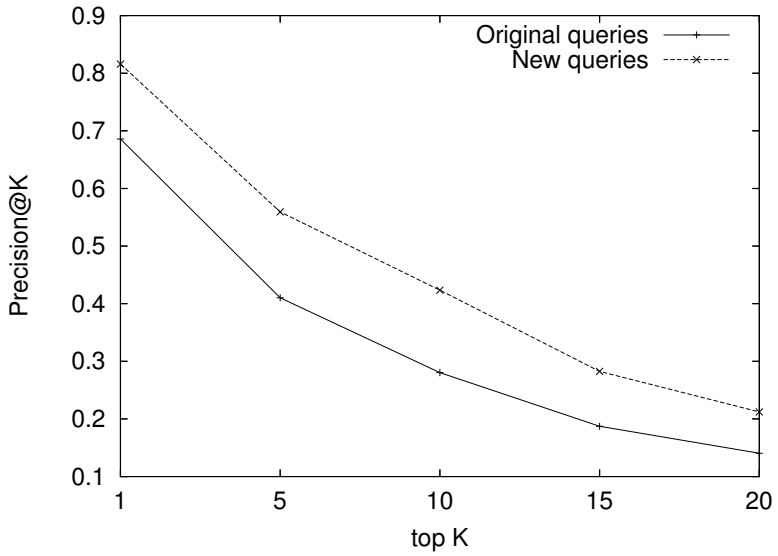

Figure 3.2: The overall performance comparison of the original and rewritten queries. We compare their results by Precision@K.

queries is about 0.28. We achieve 41.3% relative improvement.

The ratio $\frac{t(w \rightarrow s|q)}{t(w \rightarrow w|q)}$ can be regarded as a confidence score of our query rewording. A high ratio means that the reworded query is more appropriate than the original query. To test this, we ordered the query pairs by the ratio in decreasing order and we then evaluate the accuracy of top m pairs by varying m from 100 to 600. Figure 3.3 shows the influence of ratio and the difference is measured by the quotient of P@10 of recommended queries over original ones. From this figure, we can see that when the ratio is higher, the performance difference is larger. This means that the ratio is a good indicator of the confidence of query rewording.

3.5.3 Term Addition Patterns

In this section, we study our context-sensitive term addition patterns and use them for query refinement.

Examples of Addition Patterns

Table 3.9 shows several examples of the mined term addition patterns and the refined queries based on Equation (3.7). For each query, all the patterns are ordered by their probabilities

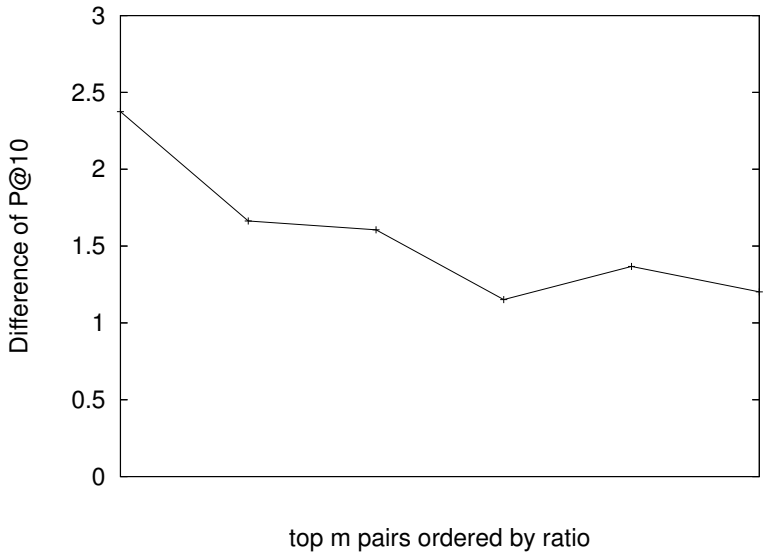

Figure 3.3: The impact of the ratio on the performance. The Difference of P@10 is measured as the quotient of P@10 of recommended queries over original ones.

in decreasing order. These examples show that our method can recommend very meaningful terms to refine an original query. For example, for the query "wedding", we can recommend meaningful terms related to different aspects of "wedding", such as "dresses" and "cakes". All these terms can help users refine their queries and thus find more coherent results. Furthermore, all these terms give good guidance for a user when he/she wants to prepare a "wedding". Such a recommendation is more useful if a user is not satisfied with the current results but lacks the necessary knowledge to think of effective words to refine his/her query.

Effectiveness Comparison

We compare our method with LLR method in a similar way as in Section 3.5.2. We use the same test set and construct our test cases similarly. The only difference is that we only retain those recommended queries which are refinements of the original queries for the LLR method. We also use 50 test cases in this experiments and use $P@5$ as the major evaluation metric. Figure 3.4 shows the comparison between different methods. In this figure, we have similar observations to the term substitution patterns: Both our and LLR methods can outperform the baseline method. Compared with LLR method, our method can achieve

q = "song lyrics"	
pattern	refined query
+christian|__song lyrics	christian song lyrics
+country|__song lyrics	country song lyrics
+gospel|__song lyrics	gospel song lyrics
+love|__song lyrics	love song lyrics
+spanish|__song lyrics	spanish song lyrics
+search|song lyrics__	song lyrics search
+worship|__song lyrics	worship song lyrics
+search|song__lyrics	song search lyrics
+praise|__song lyrics	praise song lyrics
+search|__song lyrics	search song lyrics
q = "baby names"	
pattern	refined query
+boy|baby__names	baby boy names
+girl|baby__names	baby girl names
+popular|__baby names	popular baby names
+meanings|baby names__	baby names meanings
+girl|__baby names	girl baby names
+unusual|__baby names	unusual baby names
+unique|__baby names	unique baby names
+irish|__baby names	irish baby names
+italian|__baby names	italian baby names
+twins|baby names__	baby names twins
q = "wedding"	
pattern	refined query
+dresses|wedding__	wedding dresses
+cakes|wedding__	wedding cakes
+invitations|wedding__	wedding invitations
+songs|wedding__	wedding songs
+favors|wedding__	wedding favors
+flowers|wedding__	wedding flowers
+gowns|wedding__	wedding gowns
+rings|wedding__	wedding rings
+toasts|wedding__	wedding toasts
+vows|wedding__	wedding vows

Table 3.9: Examples of term addition patterns. All the patterns are ordered according to their probabilities in decreasing order.

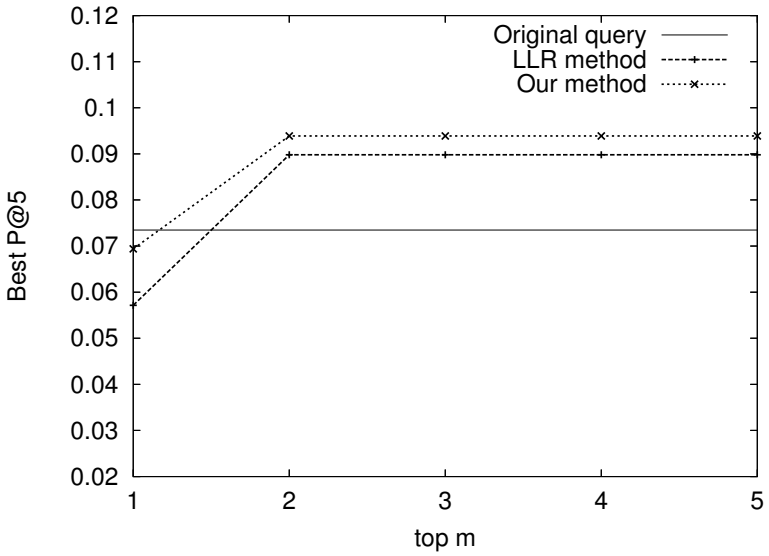

Figure 3.4: Comparison with the LLR method of term addition patterns.

better results. This is because LLR method only consider the query refinement within user sessions. Our method can utilize information across sessions since our contextual models are built over the whole collection.

3.5.4 Implementation and Efficiency

The efficiency of the algorithm is quite important since a query collection is huge. We test the efficiency of our method in this section.

We implement our algorithm using the Lemur toolkit[1]. The original data is a collection of queries. We build the standard Lemur index by treating each query as a document. This part is as efficient as the standard document indexing, thus it can be applied to very large data set pretty easily. The basic knowledge we discovered from the query collection includes the contextual models and translation models. Both are processed offline based on the Lemur index of the query collection. Based on the translation models and contextual models, the context-sensitive term substitution and term addition patterns are discovered in an online manner. Once the knowledge is built, the online part needs only fetch the corresponding knowledge we stored, and thus can be quite efficient. Since all the online parts are very efficient, in the following, we only test the efficiency of the offline part which

[1]http://www.lemurproject.org/

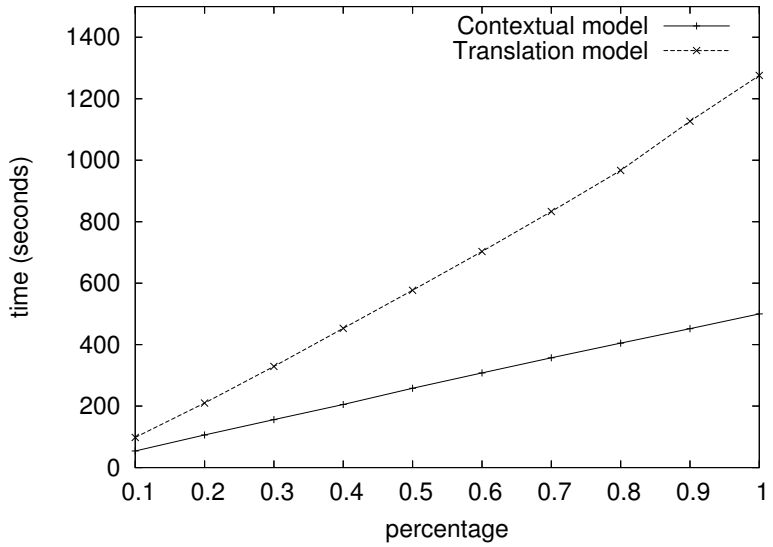

Figure 3.5: The time complexity of building the contextual models and translation models.

is to compute the contextual models and translation models.

To study the efficiency and scalability, we randomly sample $f\%$ of the original queries from the whole query collection. We vary f from 10 to 100 with step 10. For each value of f, we record the time needed for our offline part. Figure 3.5 shows the time complexity of our algorithms. In this figure, x-axis is the value of $f\%$ and y-axis is the time. It can be seen that both lines are roughly linear and thus our offline part is linearly scalable. This shows that our proposed methods can mine those patterns very efficiently and can be applicable to very large query collections.

3.6 Conclusions and Future Work

In the chapter, we studied the problem of mining term association patterns from the vast amount of search engine log data. We defined two novel term association patterns (i.e., context-sensitive term substitution and term addition patterns) and proposed new methods for mining such patterns from search engine logs. Our methods are based on the contextual and translation models which are mined from a query collection. The two types of discovered term association patterns can be used to address the mis-specification and under-specification problems of ineffective queries. Experiment results on search engine

logs show the effectiveness of our proposed methods.

There are a few limitations of our work. First, all the experiments are based on click-throughs instead of real relevance judgments, so an interesting future work would be to further test the proposed methods with real relevance judgments. Second, building an interactive user interface which can allow a user to modify his/her queries using our suggested terms can help evaluate our algorithms. Third, search logs have more meaningful click-through information besides queries and sessions. We can extend our pattern mining algorithms to incorporate this click-through information in the future.

Chapter 4

Search Result Organization

4.1 Introduction

After a user submits a query, the next step in the search cycle is for the user to examine the search results. Effective organization of search results is critical in this stage. For difficult queries, especially ambiguous queries and queries lack of discrimination, the returned results may mix several subtopics together and the user interested results may be far down in the ranked list. Partitioning search results into user interested subtopics can improve the utility of search engines for these difficult queries.

Indeed, the utility of a search engine is affected by multiple factors. While the primary factor is the soundness of the underlying retrieval model and ranking function, how to organize and present search results is also a very important factor that can affect the utility of a search engine significantly. Compared with the vast amount of literature on retrieval models, however, there is relatively little research on how to improve the effectiveness of search result organization.

The most common strategy of presenting search results is a simple ranked list. Intuitively, such a presentation strategy is reasonable for non-ambiguous, homogeneous search results; in general, it would work well when the search results are good and a user can easily find many relevant documents in the top ranked results.

However, when the search results are diverse (e.g., due to ambiguity or multiple aspects of a topic) as is often the case in Web search, the ranked list presentation would not be effective; in such a case, it would be better to group the search results into clusters so that a user can easily navigate into a particular interesting group. For example, the results in the first page returned from Google for the ambiguous query "jaguar" (as of Dec. 2nd,

2006) contain at least four different senses of "jaguar" (i.e., car, animal, software, and a sports team); even for a more refined query such as "jaguar team picture", the results are still quite ambiguous, including at least four different jaguar teams – a wrestling team, a jaguar car team, Southwestern College Jaguar softball team, and Jacksonville Jaguar football team. Moreover, if a user wants to find a place to download a jaguar software, a query such as "download jaguar" is also not very effective as the dominating results are about downloading jaguar brochure, jaguar wallpaper, and jaguar DVD. In these examples, a clustering view of the search results would be much more useful to a user than a simple ranked list. Clustering is also useful when the search results are poor, in which case, a user would otherwise have to go through a long list sequentially to reach the very first relevant document.

As a primary alternative strategy for presenting search results, clustering search results has been studied relatively extensively [29, 47, 86, 87, 88]. The general idea in virtually all the existing work is to perform clustering on a set of top-ranked search results to partition the results into *natural* clusters, which often correspond to different subtopics of the general query topic. A label will be generated to indicate what each cluster is about. A user can then view the labels to decide which cluster to look into. Such a strategy has been shown to be more useful than the simple ranked list presentation in several studies [19, 29, 86].

However, this clustering strategy has two deficiencies which make it not always work well:

First, the clusters discovered in this way do not necessarily correspond to the interesting aspects of a topic from the user's perspective. For example, users are often interested in finding either "phone codes" or "zip codes" when entering the query "area codes." But the clusters discovered by the current methods may partition the results into "local codes" and "international codes." Such clusters would not be very useful for users; even the best cluster would still have a low precision.

Second, the cluster labels generated are not informative enough to allow a user to identify the right cluster. There are two reasons for this problem: (1) The clusters are not corresponding to a user's interests, so their labels would not be very meaningful or

useful. (2) Even if a cluster really corresponds to an interesting aspect of the topic, the label may not be informative because it is usually generated based on the contents in a cluster, and it is possible that the user is not very familiar with some of the terms. For example, the ambiguous query "jaguar" may mean an animal or a car. A cluster may be labeled as "panthera onca." Although this is an accurate label for a cluster with the "animal" sense of "jaguar", if a user is not familiar with the phrase, the label would not be helpful.

In this chapter, we propose a different strategy for partitioning search results, which addresses these two deficiencies through imposing a user-oriented partitioning of the search results. That is, we try to figure out what aspects of a search topic are likely interesting to a user and organize the results accordingly. Specifically, we propose to do the following:

First, we will learn "interesting aspects" of similar topics from search logs and organize search results based on these "interesting aspects". For example, if the current query has occurred many times in the search logs, we can look at what kinds of pages viewed by the users in the results and what kind of words are used together with such a query. In case when the query is ambiguous such as "jaguar" we can expect to see some clear clusters corresponding different senses of "jaguar". More importantly, even if a word is not ambiguous (e.g., "car"), we may still discover interesting aspects such as "car rental" and "car pricing" (which happened to be the two primary aspects discovered in our search log data). Such aspects can be very useful for organizing future search results about "car". Note that in the case of "car", clusters generated using regular clustering may not necessarily reflect such interesting aspects about "car" from a user's perspective, even though the generated clusters are coherent and meaningful in other ways.

Second, we will generate more meaningful cluster labels using past query words entered by users. Assuming that the past search logs can help us learn what specific aspects are interesting to users given the current query topic, we could also expect that those query words entered by users in the past that are associated with the current query can provide meaningful descriptions of the distinct aspects. Thus they can be better labels than those extracted from the ordinary contents of search results.

ID	Query	URL	Time
1	win zip	http://www.winzip.com	xxxx
1	win zip	http://www.swinzip.com/winzip	xxxx
2	time zones	http://www.timeanddate.com	xxxx
...	...	...	...

Table 4.1: Sample entries of search engine logs. Different ID's mean different sessions.

To implement the ideas presented above, we rely on search engine logs and build a history collection containing the past queries and the associated clickthroughs. Given a new query, we find its related past queries from the history collection and learn aspects through applying the star clustering algorithm [3] to these past queries and clickthroughs. We can then organize the search results into these aspects using categorization techniques and label each aspect by the most representative past query in the query cluster.

We evaluate our method for result organization using logs of a commercial search engine. We compare our method with the default search engine ranking and the traditional clustering of search results. The results show that our method is effective for improving search utility and the labels generated using past query words are more readable than those generated using traditional clustering approaches.

The rest of the chapter is organized as follows. In Section 4.2, we describe search engine log data and our procedure of building a history collection. In Section 4.3, we present our approach in details. We describe the data set in Section 4.4 and the experimental results are discussed in Section 4.5. Finally, we conclude this chapter and discuss future work in Section 4.6.

4.2 Search Engine Logs

Search engine logs record the activities of Web users, which reflect the actual users' needs or interests when conducting Web search. They generally have the following information: text queries that users submitted, the URLs that they clicked after submitting the queries, and the time when they clicked. Search engine logs are separated by *sessions*. A session includes a single query and all the URLs that a user clicked after issuing the query [74]. A small sample of search log data is shown in Table 4.1.

Our idea of using search engine logs is to treat these logs as past history, learn users' interests using this history data automatically, and represent their interests by representative queries. For example, in the search logs, a lot of queries are related to "car" and this reflects that a large number of users are interested in information about "car." Different users are probably interested in different aspects of "car." Some are looking for renting a car, thus may submit a query like "car rental"; some are more interested in buying a used car, and may submit a query like "used car"; and others may care more about buying a car accessory, so they may use a query like "car audio." By mining all the queries which are related to the concept of "car", we can learn the aspects that are likely interesting from a user's perspective. As an example, the following is some aspects about "car" learned from our search log data (see Section 4.4).

1. `car rental, hertz car rental, enterprise car rental, ...`
2. `car pricing, used car, car values, ...`
3. `car accidents, car crash, car wrecks, ...`
4. `car audio, car stereo, car speaker, ...`

In order to learn aspects from search engine logs, we preprocess the raw logs to build a history data collection. As shown above, search engine logs consist of sessions. Each session contains the information of the text query and the clicked Web page URLs, together with the time that the user did the clicks. However, this information is limited since URLs alone are not informative enough to tell the intended meaning of a submitted query accurately. To gather rich information, we enrich each URL with additional text content. Specifically, given the query in a session, we obtain its top-ranked results using the search engine from which we obtained our log data, and extract the snippets of the URLs that are clicked on according to the log information in the corresponding session. All the titles, snippets, and URLs of the clicked Web pages of that query are used to represent the session.

Different sessions may contain the same queries. Thus the number of sessions could be quite huge and the information in the sessions with the same queries could be redundant. In order to improve the scalability and reduce data sparseness, we aggregate all the sessions

which contain exactly the same queries together. That is, for each unique query, we build a "pseudo-document" which consists of all the descriptions of its clicks in all the sessions aggregated. The keywords contained in the queries themselves can be regarded as brief summaries of the pseudo-documents. All these pseudo-documents form our history data collection, which is used to learn interesting aspects in the following section.

4.3 Our Approach

Our approach is to organize search results by aspects learned from search engine logs. Given an input query, the general procedure of our approach is:

1. Get its related information from search engine logs. All the information forms a working set.

2. Learn aspects from the information in the working set. These aspects correspond to users' interests given the input query. Each aspect is labeled with a representative query.

3. Categorize and organize the search results of the input query according to the aspects learned above.

We now give a detailed presentation of each step.

4.3.1 Finding Related Past Queries

Given a query q, a search engine will return a ranked list of Web pages. To know what the users are really interested in given this query, we first retrieve its past similar queries in our preprocessed history data collection.

Formally, assume we have N pseudo-documents in our history data set: $H = \{Q_1, Q_2, ..., Q_N\}$. Each Q_i corresponds to a unique query and is enriched with clickthrough information as discussed in Section 4.2. To find q's related queries in H, a natural way is to use a text retrieval algorithm. Here we use the OKAPI method [51], one of the state-of-the-art retrieval methods. Specifically, we use the following formula to calculate the similarity between query

q and pseudo-document Q_i:

$$\sum_{w \in q \bigcap Q_i} c(w,q) \times IDF(w) \times \frac{(k_1+1) \times c(w,Q_i)}{k_1((1-b)+b\frac{|Q_i|}{avdl}) + c(w,Q_i)}$$

where k_1 and b are OKAPI parameters set empirically, $c(w,Q_i)$ and $c(w,q)$ are the count of word w in Q_i and q respectively, $IDF(w)$ is the inverse document frequency of word w, and $avdl$ is the average document length in our history collection.

Based on the similarity scores, we rank all the documents in H. The top ranked documents provide us a working set to learn the aspects that users are usually interested in. Each document in H corresponds to a past query, and thus the top ranked documents correspond to q's related past queries.

4.3.2 Learning Aspects by Clustering

Given a query q, we use $H_q = \{d_1, ..., d_n\}$ to represent the top ranked pseudo-documents from the history collection H. These pseudo-documents contain the aspects that users are interested in. In this subsection, we propose to use a clustering method to discover these aspects.

Any clustering algorithm could be applied here. In this chapter, we use an algorithm based on graph partition: the star clustering algorithm [3]. A good property of the star clustering in our setting is that it can suggest a good label for each cluster naturally. We describe the star clustering algorithm below.

Star Clustering

Given H_q, star clustering starts with constructing a pair-wise similarity graph on this collection based on the vector space model in information retrieval [56]. Then the clusters are formed by dense subgraphs that are star-shaped. These clusters form a cover of the similarity graph. Formally, for each of the n pseudo-documents $\{d_1, ..., d_n\}$ in the collection H_q, we compute a TF-IDF vector. Then, for each pair of documents d_i and d_j $(i \neq j)$, their

similarity is computed as the cosine score of their corresponding vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, that is

$$sim(d_i, d_j) = \cos(\mathbf{v}_i, \mathbf{v}_j) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{|\mathbf{v}_i| \cdot |\mathbf{v}_j|}.$$

A similarity graph G_σ can then be constructed as follows using a similarity threshold parameter σ. Each document d_i is a vertex of G_σ. If $sim(d_i, d_j) > \sigma$, there would be an edge connecting the corresponding two vertices. After the similarity graph G_σ is built, the star clustering algorithm clusters the documents using a greedy algorithm as follows:

1. Associate every vertex in G_σ with a flag, initialized as *unmarked*.
2. From those *unmarked* vertices, find the one which has the highest degree and let it be u.
3. Mark the flag of u as *center*.
4. Form a cluster C containing u and all its neighbors that are not marked as *center*. Mark all the selected neighbors as *satellites*.
5. Repeat from step 2 until all the vertices in G_σ are marked.

Each cluster is *star-shaped*, which consists a single *center* and several *satellites*. There is only one parameter σ in the star clustering algorithm. A big σ enforces that the connected documents have high similarities, and thus the clusters tend to be small. On the other hand, a small σ will make the clusters big and less coherent. We will study the impact of this parameter in our experiments.

A good feature of the star clustering algorithm is that it outputs a *center* for each cluster. In the past query collection H_q, each document corresponds to a query. This *center* query can be regarded as the most representative one for the whole cluster, and thus provides a *label* for the cluster naturally. All the clusters obtained are related to the input query q from different perspectives, and they represent the possible aspects of interests about query q of users.

4.3.3 Categorizing Search Results

In order to organize the search results according to users' interests, we use the learned aspects from the related past queries to categorize the search results. Given the top m Web pages returned by a search engine for q: $\{s_1, ..., s_m\}$, we group them into different aspects using a categorization algorithm.

In principle, any categorization algorithm can be used here. Here we use a simple centroid-based method for categorization. Naturally, more sophisticated methods such as SVM [65] may be expected to achieve even better performance.

Based on the pseudo-documents in each discovered aspect C_i, we build a centroid prototype $\mathbf{p}_i$ by taking the average of all the vectors of the documents in C_i:

$$\mathbf{p}_i = \frac{1}{|C_i|} \sum_{l \in C_i} \mathbf{v}_l.$$

All these $\mathbf{p}_i$'s are used to categorize the search results. Specifically, for any search result s_j, we build a TF-IDF vector. The centroid-based method computes the cosine similarity between the vector representation of s_j and each centroid prototype $\mathbf{p}_i$. We then assign s_j to the aspect with which it has the highest cosine similarity score.

All the aspects are finally ranked according to the number of search results they have. Within each aspect, the search results are ranked according to their original search engine ranking.

4.4 Data Collection

We construct our data set based on the MSN search log data set released by the Microsoft Live Labs in 2006 [42]. In total, this log data spans 31 days from 05/01/2006 to 05/31/2006. There are 8,144,000 queries, 3,441,000 distinct queries, and 4,649,000 distinct URLs in the raw data.

To test our algorithm, we separate the whole data set into two parts according to the time: the first 2/3 data is used to simulate the historical data that a search engine accumulated, and we use the last 1/3 to simulate future queries. In the history collection,

we clean the data by only keeping those frequent, well-formatted, English queries (queries which only contain characters 'a', 'b', ..., 'z', and space, and appear more than 5 times). After cleaning, we get 169,057 unique queries in our history data collection totally. On average, each query has 3.5 distinct clicks. We build the "pseudo-documents" for all these queries as described in Section 4.2. The average length of these pseudo-documents is 68 words and the total data size of our history collection is 129MB.

We construct our test data from the last 1/3 data. According to the time, we separate this data into two test sets equally for cross-validation to set parameters. For each test set, we use every session as a test case. Each session contains a single query and several clicks. (Note that we do not aggregate sessions for test cases. Different test cases may have the same queries but possibly different clicks.) Since it is infeasible to ask the original user who submitted a query to judge the results for the query, we follow the work [33] and opt to use the clicks associated with the query in a session to *approximate* relevant documents. Using clicks as judgments, we can then compare different algorithms for organizing search results to see how well these algorithms can help users reach the clicked URLs.

Organizing search results into different aspects is expected to help informational queries. It thus makes sense to focus on the informational queries in our evaluation. For each test case, i.e., each session, we count the number of different clicks and filter out those test cases with fewer than 4 clicks under the assumption that a query with more clicks is more likely to be an informational query. Since we want to test whether our algorithm can learn from the past queries, we also filter out those test cases whose queries can not retrieve at least 100 pseudo-documents from our history collection. Finally, we obtain 172 and 177 test cases in the first and second test sets respectively. On average, we have 6.23 and 5.89 clicks for each test case in the two test sets respectively.

4.5 Experiments

In the section, we describe our experiments on the search result organization based past search engine logs.

4.5.1 Experimental Design

We use two baseline methods to evaluate the proposed method for organizing search results. For each test case, the first method is the default ranked list from a search engine (baseline). The second method is to organize the search results by clustering them (cluster-based). For fair comparison, we use the same clustering algorithm as our log-based method (i.e., star clustering). That is, we treat each search result as a document, construct the similarity graph, and find the star-shaped clusters. We compare our method (log-based) with the two baseline methods in the following experiments. For both cluster-based and log-based methods, the search results within each cluster is ranked based on their original ranking given by the search engine.

To compare different result organization methods, we adopt a similar method as in the paper [29]. That is, we compare the quality (e.g., precision) of the best cluster, which is defined as the one with the largest number of relevant documents. Organizing search results into clusters is to help users navigate into relevant documents quickly. The above metric is to simulate a scenario when users always choose the right cluster and look into it. Specifically, we download and organize the top 100 search results into aspects for each test case. We use Precision at 5 documents (P@5) in the best cluster as the primary measure to compare different methods. P@5 is a very meaningful measure as it tells us the *perceived* precision when the user opens a cluster and looks at the first 5 documents. We also use Mean Reciprocal Rank (MRR) as another metric. MRR is calculated as

$$MRR = \frac{1}{|T|} \sum_{q \in T} \frac{1}{r_q}$$

where T is a set of test queries, r_q is the rank of the first relevant document for q.

To give a fair comparison across different organization algorithms, we force both cluster-based and log-based methods to output the same number of aspects and force each search result to be in one and only one aspect. The number of aspects is fixed at 10 in all the following experiments. The star clustering algorithm can output different number of clusters for different input. To constrain the number of clusters to 10, we order all the clusters by

Method	Test set 1		Test set 2	
	MRR	P@5	MRR	P@5
Baseline	0.7347	0.3325	0.7393	0.3288
Cluster-based	0.7735	0.3162	0.7666	0.2994
Log-based	0.7833	0.3534	0.7697	0.3389
Cluster/Baseline	5.28%	-4.87%	3.69%	-8.93%
Log/Baseline	6.62%	6.31%	4.10%	3.09%
Log/Cluster	1.27%	11.76%	0.40%	13.20%

Table 4.2: Comparison of different methods by MMR and P@5. We also show the percentage of relative improvement in the lower part.

Comparison	Test set 1	Test set 2
	Impr./Decr.	Impr./Decr.
Cluster/Baseline	53/55	50/64
Log/Baseline	55/44	60/45
Log/Cluster	68/47	69/44

Table 4.3: Pairwise comparison w.r.t the number of test cases whose P@5's are improved versus decreased w.r.t the baseline.

their sizes, select the top 10 as aspect candidates. We then re-assign each search result to one of these selected 10 aspects that has the highest similarity score with the corresponding aspect centroid. In our experiments, we observe that the sizes of the best clusters are all larger than 5, and this ensures that P@5 is a meaningful metric.

4.5.2 Experimental Results

Our main hypothesis is that organizing search results based on the users' interests learned from a search log data set is more beneficial than to organize results using a simple list or cluster search results. In the following, we test our hypothesis from two perspectives – organization and labeling.

Overall performance

We compare three methods, basic search engine ranking (baseline), traditional clustering based method (cluster-based), and our log based method (log-based), in Table 4.2 using MRR and P@5. We optimize the parameter σ's for each collection individually based on P@5 values. This shows the best performance that each method can achieve. In this table,

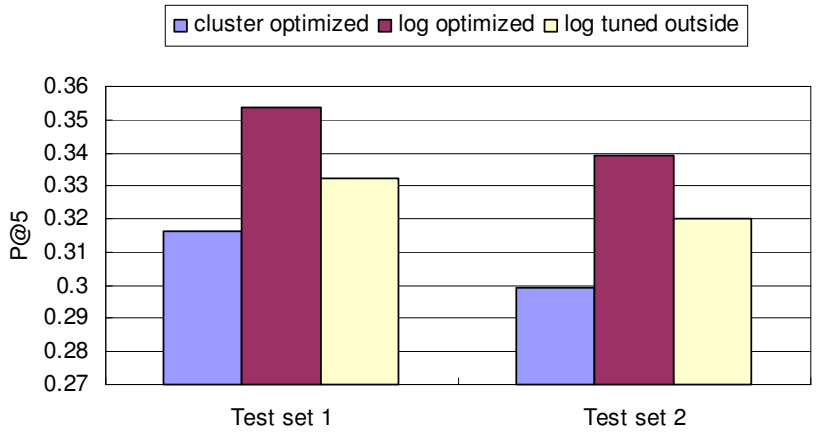

Figure 4.1: Results using parameters tuned from the other test collection. We compare it with the optimal performance of the cluster-based and our log-based methods.

we can see that in both test collections, our method is better than both the "baseline" and the "cluster-based" methods. For example, in the first test collection, the baseline method of MMR is 0.734, the cluster-based method is 0.773 and our method is 0.783. We achieve higher accuracy than both cluster-based method (1.27% improvement) and the baseline method (6.62% improvement). The P@5 values are 0.332 for the baseline, 0.316 for cluster-based method, but 0.353 for our method. Our method improves over the baseline by 6.31%, while the cluster-based method even decreases the accuracy. This is because cluster-based method organizes the search results only based on the contents. Thus it could organize the results differently from users' preferences. This confirms our hypothesis of the bias of the cluster-based method. Comparing our method with the cluster-based method, we achieve significant improvement on both test collections. The p-values of the significance tests based on P@5 on both collections are 0.01 and 0.02 respectively. This shows that our log-based method is effective to learn users' preferences from the past query history, and thus it can organize the search results in a more useful way to users.

We showed the optimal results above. To test the sensitivity of the parameter σ of our log-based method, we use one of the test sets to tune the parameter to be optimal and then use the tuned parameter on the other set. We compare this result (log tuned outside) with the optimal results of both cluster-based (cluster optimized) and log-based methods (log optimized) in Figure 4.1. We can see that, as expected, the performance using the

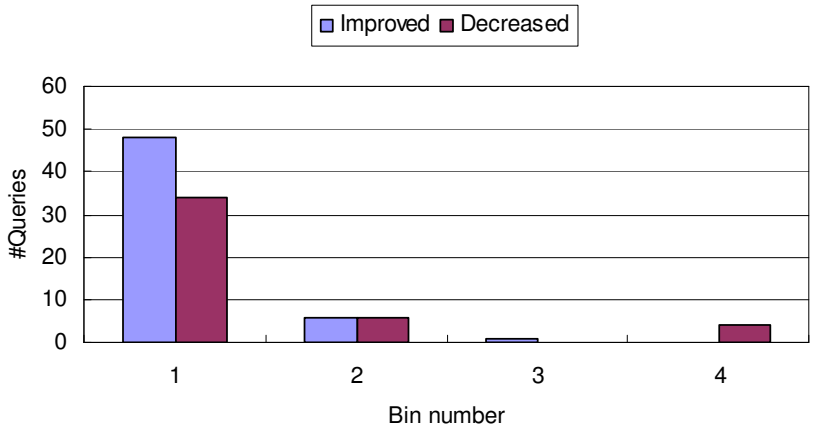

Figure 4.2: The correlation between performance change and result diversity.

parameter tuned on a separate set is worse than the optimal performance. However, our method still performs much better than the *optimal* results of cluster-based method on both test collections.

In Table 4.3, we show pairwise comparisons of the three methods in terms of the numbers of test cases for which P@5 is increased versus decreased. We can see that our method improves more test cases compared with the other two methods. In the next section, we show more detailed analysis to see what types of test cases can be improved by our method.

Detailed Analysis

To better understand the cases where our log-based method can improve the accuracy, we test two properties: result diversity and query difficulty. All the analysis below is based on test set 1.

Diversity Analysis: Intuitively, organizing search results into different aspects is more beneficial to those queries whose results are more diverse, as for such queries, the results tend to form two or more big clusters. In order to test the hypothesis that log-based method help more those queries with diverse results, we compute the size ratios of the biggest and second biggest clusters in our log-based results and use this ratio as an indicator of diversity. If the ratio is small, it means that the first two clusters have a small difference thus the results are more diverse. In this case, we would expect our method to help more. The results are shown in Figure 4.2. In this figure, we partition the ratios into 4 bins. The

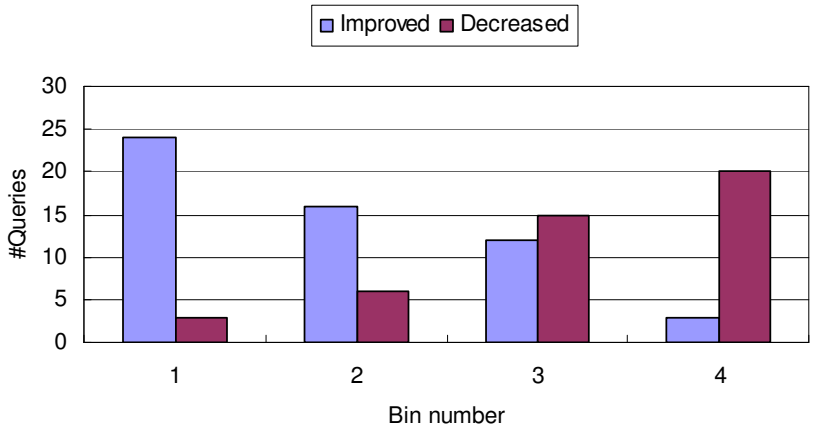

Figure 4.3: The correlation between performance change and query difficulty.

4 bins correspond to the ratio ranges $[1, 2)$, $[2, 3)$, $[3, 4)$, and $[4, +\infty)$ respectively. ($[i, j)$ means that $i \leq$ ratio $< j$.) In each bin, we count the numbers of test cases whose P@5's are improved versus decreased with respect to the ranking baseline, and plot the numbers in this figure. We can observe that when the ratio is smaller, the log-based method can improve more test cases. But when the ratio is large, the log-based method can not improve over the baseline. For example, in bin 1, 48 test cases are improved and 34 are decreased. But in bin 4, all the 4 test cases are decreased. This confirms our hypothesis that our method can help more if the query has more diverse results. This also suggests that we should "turn off" the option of re-organizing search results if the results are not very diverse (e.g., as indicated by the cluster size ratio).

Difficulty Analysis: Difficult queries have been studied in recent years [14, 84, 11]. Here we analyze the effectiveness of our method in helping difficult queries. We quantify the query difficulty by the Mean Average Precision (MAP) of the original search engine ranking for each test case. We then order the 172 test cases in test set 1 in an increasing order of MAP values. We partition the test cases into 4 bins with each having a roughly equal number of test cases. A small MAP means that the utility of the original ranking is low. Bin 1 contains those test cases with the lowest MAP's and bin 4 contains those test cases with the highest MAP's. For each bin, we compute the numbers of test cases whose P@5's are improved versus decreased. Figure 4.3 shows the results. Clearly, in bin 1, most of the test cases are improved (24 vs 3), while in bin 4, log-based method may decrease the

performance (3 vs 20). This shows that our method is more beneficial to difficult queries, which is as expected since clustering search results is intended to help difficult queries. This also shows that our method does not really help easy queries, thus we should turn off our organization option for easy queries.

Parameter Setting

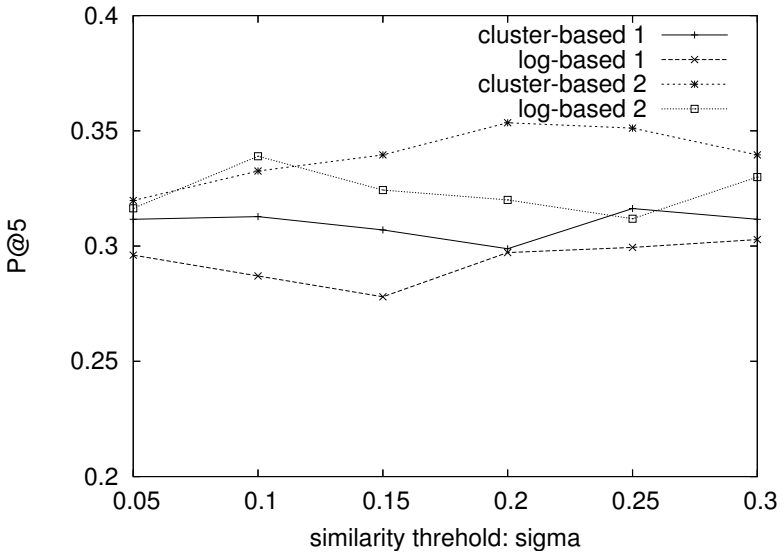

Figure 4.4: The impact of similarity threshold σ on both cluster-based and log-based methods. We show the result on both test collections.

We examine parameter sensitivity in this section. For the star clustering algorithm, we study the similarity threshold parameter σ. For the OKAPI retrieval function, we study the parameters k_1 and b. We also study the impact of the number of past queries retrieved in our log-based method.

Figure 4.4 shows the impact of the parameter σ for both cluster-based and log-based methods on both test sets. We vary σ from 0.05 to 0.3 with step 0.05. Figure 4.4 shows that the performance is not very sensitive to the parameter σ. We can always obtain the best result in range $0.1 \leq \sigma \leq 0.25$.

In Table 4.4, we show the impact of OKAPI parameters. We vary k_1 from 1.0 to 2.0 with step 0.2 and b from 0 to 1 with step 0.2. From this table, it is clear that P@5 is also not very sensitive to the parameter setting. Most of the values are larger than 0.35. The

		b					
		0.0	0.2	0.4	0.6	0.8	1.0
k_1	1.0	0.3476	0.3406	0.3453	0.3616	0.3500	0.3453
	1.2	0.3418	0.3383	0.3453	0.3593	0.3534	0.3546
	1.4	0.3337	0.3430	0.3476	0.3604	0.3546	0.3465
	1.6	0.3476	0.3418	0.3523	0.3534	0.3581	0.3476
	1.8	0.3465	0.3418	0.3546	0.3558	0.3616	0.3476
	2.0	0.3453	0.3500	0.3534	0.3558	0.3569	0.3546

Table 4.4: Impact of OKAPI parameters k_1 and b.

default values $k_1 = 1.2$ and $b = 0.8$ give approximately optimal results.

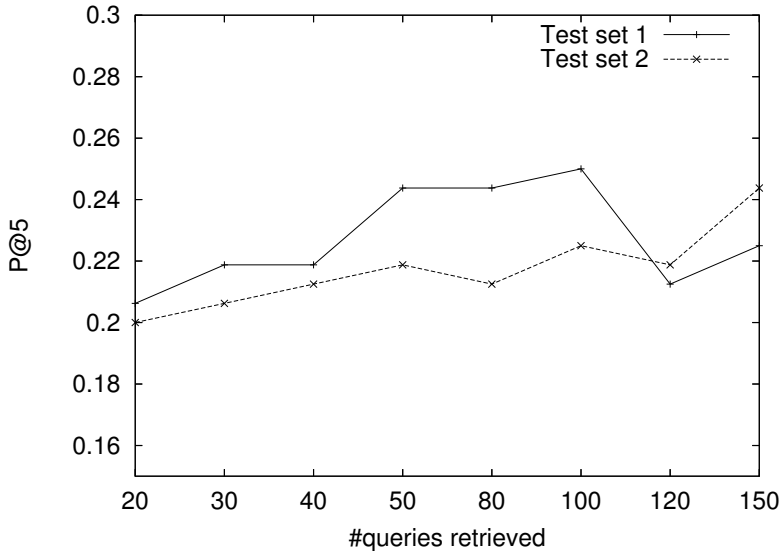

Figure 4.5: The impact of the number of past queries retrieved.

We further study the impact of the amount of history information to learn from by varying the number of past queries to be retrieved for learning aspects. The results on both test collections are shown in Figure 4.5. We can see that the performance gradually increases as we enlarge the number of past queries retrieved. Thus our method could potentially learn more as we accumulate more history. More importantly, as time goes, more and more queries will have sufficient history, so we can improve more and more queries.

An Illustrative Example

We use the query "area codes" to show the difference in the results of the log-based method and the cluster-based method. This query may mean "phone codes" or "zip codes". Ta-

Cluster-based method	Log-based method
city, state	telephone, city, international
local, area	phone, dialing
international	zip, postal

Table 4.5: An example showing the difference between the cluster-based method and our log-based method

ble 4.5 shows the representative keywords extracted from the three biggest clusters of both methods. In the cluster-based method, the results are partitioned based on locations: local or international. In the log-based method, the results are disambiguated into two senses: "phone codes" or "zip codes". While both are reasonable partitions, our evaluation indicates that most users using such a query are often interested in either "phone codes" or "zip codes." since the P@5 values of cluster-based and log-based methods are 0.2 and 0.6, respectively. Therefore our log-based method is more effective in helping users to navigate into their desired results.

Labeling Comparison

We now compare the labels between the cluster-based method and log-based method. The cluster-based method has to rely on the keywords extracted from the snippets to construct the label for each cluster. Our log-based method can avoid this difficulty by taking advantage of queries. Specifically, for the cluster-based method, we count the frequency of a keyword appearing in a cluster and use the most frequent keywords as the cluster label. For log-based method, we use the *center* of each star cluster as the label for the corresponding cluster.

In general, it is not easy to quantify the readability of a cluster label automatically. We use examples to show the difference between the cluster-based and the log-based methods. In Table 4.6, we list the labels of the top 5 clusters for two examples "jaguar" and "apple". For the cluster-based method, we separate keywords by commas since they do not form a phrase. From this table, we can see that our log-based method gives more readable labels because it generates labels based on users' queries. This is another advantage of our way of organizing search results over the clustering approach.

Label comparison for query "jaguar"			
Log-based method		Cluster-based method	
1.	jaguar animal	1.	jaguar, auto, accessories
2.	jaguar auto accessories	2.	jaguar, type, prices
3.	jaguar cats	3.	jaguar, panthera, cats
4.	jaguar repair	4.	jaguar, services, boston
5.	jaguar animal pictures	5.	jaguar, collection, apparel
Label comparison for query "apple"			
Log-based method		Cluster-based method	
1.	apple computer	1.	apple, support, product
2.	apple ipod	2.	apple, site, computer
3.	apple crisp recipe	3.	apple, world, visit
4.	fresh apple cake	4.	apple, ipod, amazon
5.	apple laptop	5.	apple, products, news

Table 4.6: Cluster label comparison.

4.6 Conclusions and Future Work

In this chapter, we studied the problem of organizing search results in a user-oriented manner. To attain this goal, we rely on search engine logs to learn interesting aspects from users' perspective. Given a query, we retrieve its related queries from past query history, learn the aspects by clustering the past queries and the associated clickthrough information, and categorize the search results into the aspects learned. We compared our log-based method with the traditional cluster-based method and the baseline of search engine ranking. The experiments show that our log-based method can consistently outperform cluster-based method and improve over the ranking baseline, especially when the queries are difficult or the search results are diverse. Furthermore, our log-based method can generate more meaningful aspect labels than the cluster labels generated based on search results when we cluster search results.

There are several interesting directions for further extending our work: First, although our experiment results have clearly shown promise of the idea of learning from search logs to organize search results, the methods we have experimented with are relatively simple. It would be interesting to explore other potentially more effective methods. In particular, we hope to develop probabilistic models for learning aspects and organizing results simultaneously. Second, with the proposed way of organizing search results, we can expect to obtain

informative feedback information from a user (e.g., the aspect chosen by a user to view). It would thus be interesting to study how to further improve the organization of the results based on such feedback information. Finally, we can combine a general search log with any personal search log to customize and optimize the organization of search results for each individual user.

Chapter 5

Negative Feedback

5.1 Introduction

A user's interactions with search results are very useful clues of the user's information need. A process which can learn from such clues to improve search accuracy is often called feedback [62]. Due to inherent limitations of current retrieval models, it is inevitable that some queries are difficult in the sense that the search results would be poor. Indeed, some queries may be so difficult that a user can not find any relevant document in a long list of top-ranked documents even if the user has reformulated the queries several times. In such a scenario, the feedback information that a user could provide, either implicitly or explicitly, is all negative. An interesting question is thus how to exploit only non-relevant information to improve search accuracy, which is referred to as *negative feedback*.

The most existing feedback techniques, such as relevance feedback [53, 31], pseudo-relevance feedback [4, 80], and implicit feedback [61], rely on positive information. In the case of a difficult topic, we likely will have only negative (i.e., non-relevant) examples, raising the important question of how to perform relevance feedback with only negative examples, i.e., negative relevance feedback. Ideally, if we can perform effective negative feedback, when the user could not find any relevant document on the first page of search results, we would be able to improve the ranking of unseen results in the next a few pages.

However, whether such negative feedback can indeed improve retrieval accuracy is still largely an open question. Indeed, the effectiveness of current feedback methods often rely on relevant documents; negative information, such as non-relevant documents, is mostly ignored in past work [20, 31].

On the surface, any standard relevance feedback technique can be applied to negative

relevance feedback. However, our recent work [69] has shown that special care and special heuristics are needed to achieve effective negative feedback. Specifically, in this work, we have shown that some language model-based feedback methods, although quite effective for exploiting positive feedback information, cannot naturally handle negative feedback, thus several methods were proposed to perform negative feedback in language modeling framework. However, this study is neither comprehensive nor conclusive for several reasons: (1) It is only limited to language models; vector space models have not been evaluated. (2) The results are evaluated over only one collection. (3) The lack of systematic experiment design and result analysis makes it hard to know the advantages or disadvantages of different methods.

In this chapter, we conduct a more systematic study of different methods for negative relevance feedback. Our study is on two representative retrieval models: vector space models and language models. We first categorize negative feedback techniques into several general strategies: single query model, single positive model with single negative query model, and single positive model with multiple negative query models. Following these strategies, we then develop a set of representative retrieval methods for both retrieval models. Systematic comparison and analysis are conducted on two large representative TREC data sets. Ideally, test sets with sufficient naturally difficult topics are required to evaluate these negative feedback methods, but there are not many naturally difficult topics in the existing TREC data collections. To overcome this difficulty, we use two sampling strategies to adapt a test collection with easy topics to evaluate negative feedback. The basic idea of our sampling methods is to simulate difficult queries from easy ones through deleting a set of relevant documents so that the results become poor. The effectiveness of these sampling methods is also verified on the TREC data sets.

Our systematic study leads to several interesting conclusions. We find that language model-based negative feedback methods are generally more effective and robust than those based on vector space models possibly due to more accurate learning of negative models. While cluster hypothesis [29] generally holds for relevant documents, our results show that negative documents do not cluster together. Thus adapting standard relevance feedback to

learn a single query model is not optimal for negative feedback, and using multiple negative models is more effective than a single negative model since negative documents may distract in different ways. Our results also show that it is feasible to adapt test collections with easy topics (through sampling) to evaluate negative feedback methods.

The rest of the chapter is organized as follows. In Section 5.2, we describe our problem setup and different techniques for negative feedback. We describe our sampling methods to simulate difficult topics by adapting easy ones in Section 5.3. Experiments are analyzed and discussed in Section 5.4. We conclude this chapter and discuss our future work in Section 5.5.

5.2 Negative Feedback Techniques

5.2.1 Problem Formulation

We formulate the problem of negative feedback in a similar way as presented in [69]. Given a query Q and a document collection $\mathcal{C}$, a retrieval system returns a ranked list of documents $\mathcal{L}$. L_i denotes the i-th ranked document in the ranked list. We assume that Q is so difficult that all the top f ranked documents (seen so far by a user) are non-relevant. The goal is to study how to use these negative examples, i.e., $\mathcal{N} = \{L_1, ..., L_f\}$, to rerank the next r unseen documents in the original ranked list: $\mathcal{U} = \{L_{f+1}, ..., L_{f+r}\}$. We set $f = 10$ to simulate that the first page of search results are irrelevant, and set $r = 1000$. We use the following notations in the rest of the chapter:

$S(Q, D)$ is the relevance score of document D for query Q.

$c(w, D)$ is the count of word w in document D.

$c(w, Q)$ is the count of word w in query Q.

$|\mathcal{C}|$ is the total number of documents in the collection $\mathcal{C}$.

$df(w)$ is the document frequency of word w.

$|D|$ is the length of document D.

$avdl$ is the average document length.

$\mathcal{N}$ is the set of negative feedback documents.

$\mathcal{U}$ is the set of unseen documents to be reranked.

5.2.2 General Strategies for Negative Feedback

Query Modification

Since negative feedback can be regarded as a special case of relevance feedback where no positive example is available, our first general strategy is simply to apply any existing feedback methods (e.g., Rocchio [53]) to use only non-relevant examples. We call this strategy *query modification* because most existing feedback methods would achieve feedback through modifying the representation of a query based on relevant and non-relevant feedback documents. In effect, they often introduce additional terms to expand a query and assign more weight to a term with more occurrences in relevant documents and less weight or negative weight to a term with more occurrences in non-relevant documents.

Some existing feedback methods, such as Rocchio method [53], already have a component for using negative information, so they can be directly applied to negative feedback. However, other methods, such as model-based feedback methods in language modeling approaches [89], can not naturally support negative feedback, thus extension has to be made to make them work for negative feedback [69]. Later we will further discuss this.

Note that with this strategy, we generally end up with one *single* query model/representation which combines both positive information from the original query and negative information from the feedback documents.

Score Combination

The query modification strategy mixes both positive and negative information together in a single query model. Sometimes it is not natural to mix these two kinds of information as in the case of using generative models for feedback [89]. A more flexible alternative strategy is to maintain a positive query representation and a negative query representation *separately*, and combine the scores of a document w.r.t. both representations. We call this strategy *score combination*.

With this strategy, negative examples can be used to learn a negative query representa-

tion which can then be used to score a document based on the likelihood that the document is a distracting non-relevant document; such a score can then be used to adjust the positive relevance score between the original query and the corresponding document.

Intuitively, a document with higher relevance score to the negative query representation can be assumed to be less relevant, thus the final score of this document can be computed as

$$S_{combined}(Q, D) = S(Q, D) - \beta \times S(Q_{neg}, D) \tag{5.1}$$

where Q_{neg} is a negative query representation and β is a parameter to control the influence of negative feedback. When $\beta = 0$, we do not perform negative feedback, and the ranking would be the same as the original ranking according to query Q. A larger value of β causes more penalization of documents similar to the negative query representation.

Equation (5.1) shows that *either* a high score of $S(Q, D)$ *or* a low score of $S(Q_{neg}, D)$ would result in a high score of $S_{combined}(Q, D)$. This means that the proposed score combination may favor non-relevant documents if they have lower similarity to the negative model; this is risky because the negative query representation is only reliable for filtering out highly similar documents. Thus a more reasonable approach would be to only penalize documents which are most similar to the negative query model and avoid affecting the relevance scores of other documents. To achieve this goal, instead of penalizing all the documents in $\mathcal{U}$, we need to penalize only a subset of documents that are most similar to the negative query. We propose to use the following two heuristics to select documents for penalization (i.e., adjusting their scores using Equation (5.1)):

Heuristic 1 (Local Neighborhood): Rank all the documents in $\mathcal{U}$ by the negative query and penalize the top ρ documents.

Heuristic 2 (Global Neighborhood): Rank all the documents in $\mathcal{C}$ by the negative query. Select, from the top ρ documents of this ranked list, those documents in $\mathcal{U}$ to penalize.

In both cases, ρ is a parameter to control the number of documents to be penalized and would be empirically set. The two heuristics essentially differ in how this ρ value affects the

number of documents in $\mathcal{U}$ to be penalized. In Heuristic 1, the actual number of documents in $\mathcal{U}$ to be penalized is fixed, i.e., ρ, but in Heuristic 2, it is dynamic and could be smaller than ρ, because the top ρ documents most similar to the negative query are generally not all in the set of $\mathcal{U}$. If we are to set ρ to a constant for all queries, intuitively Heuristic 2 can be more robust than Heuristics 1, which is confirmed in our experiments.

How do we compute the negative query representation Q_neg and the score $S(Q_{neg}, D)$? A simple strategy is to combine all the negative information from $\mathcal{N}$ to form a single negative query representation, which would be referred to as "Single Negative Model" (*SingleNeg*). However, unlike positive information, negative information might be quite diverse. Thus, it is more desirable to capture negative information with more than one negative query model. Formally, let Q^i_{neg}, where $1 \leq i \leq k$, be k negative query models, we may compute $S(Q_{neg}, D)$ as follows:

$$S(Q_{neg}, D) = F(\bigcup_{i=1}^{k} \{S(Q^i_{neg}, D)\})$$

where F is an aggregation function to combine the set of k values. We call this method "Multiple Negative Models" (*MultiNeg*).

Summary

We have discussed two general strategies with some variations for negative feedback, which can be summarized as follows: (1) **SingleQuery**: query modification strategy; (2) **SingleNeg**: score combination with a single negative query model; (3) **MultiNeg**: score combination with multiple negative query models. For both *SingleNeg* and *MultiNeg* models, we can use either of the two heuristics proposed in the previous subsection to penalize documents selectively. In the next subsection, we discuss some specific ways of implementing these general strategies in both vector space models and language models:

5.2.3 Negative Feedback in Vector Space Model

In vector space models, documents and queries are represented as vectors in a high-dimensional space spanned by terms. The weight of a term w in document D can be computed in many different ways and typically a similar measure such as dot product is used to score

documents[62].

In our experiments, we use the following BM25 weight [52]:

$$\frac{(k_1+1)\times c(w,D)}{k_1((1-b)+b\frac{|D|}{avdl})+c(w,D)}\times\log(\frac{|\mathcal{C}|+1}{df(w)}) \tag{5.2}$$

where k_1 and b are parameters. The weight of a query term is set to the raw term frequency, i.e., $c(w,Q)$. We compute the relevance score using the dot product: $S(Q,D)=\overrightarrow{Q}\cdot\overrightarrow{D}$ where $\overrightarrow{D}$ and $\overrightarrow{Q}$ represent document vector and query vector, respectively.

SingleQuery Strategy

The Rocchio method [53] is a commonly used feedback method in vector space models. The idea is to update a query vector with both relevant and non-relevant documents. When only non-relevant documents $\mathcal{N}$ are available, the Rocchio method can be written as

$$\overrightarrow{Q}_{new}=\overrightarrow{Q}-\gamma\times\frac{1}{|\mathcal{N}|}\sum_{D\in\mathcal{N}}\overrightarrow{D}. \tag{5.3}$$

This gives us an updated query vector $\overrightarrow{Q}_{new}$, which can be used to rerank documents in $\mathcal{U}$.

SingleNeg Strategy

SingleNeg adjusts the original relevance score of a document with a single negative query. We compute the negative query as the center of negative documents, i.e., $\overrightarrow{Q}_{neg}=\frac{1}{|\mathcal{N}|}\sum_{D\in\mathcal{N}}\overrightarrow{D}$. Using Equation (5.1), the combined score of a document D is

$$S_{combined}(Q,D)=\overrightarrow{Q}\cdot\overrightarrow{D}-\beta\times\overrightarrow{Q}_{neg}\cdot\overrightarrow{D}. \tag{5.4}$$

It is straightforward to verify that Equation (5.3) and (5.4) are equivalent. However, SingleNeg has the advantage of allowing us to penalize negative documents selectively using either of the two heuristics presented earlier.

MultiNeg Strategy

MultiNeg adjusts the original relevance score of a document with multiple negative queries which can be obtained, e.g., through clustering. In our experiments, we take each negative document as a negative query and use max as our aggregation function. Intuitively, max allows us to penalize any document that is close to at least one negative document. Thus

$$S(Q_{neg}, D) = \max(\bigcup_{Q' \in \mathcal{N}} \{\overrightarrow{Q'} \cdot \overrightarrow{D}\}).$$

This score is then combined with $S(Q, D)$ to rerank the documents in $\mathcal{U}$. Again, we have two variants of this method corresponding to applying the two heuristics discussed above.

5.2.4 Negative Feedback for Language Models

KL-divergence retrieval model [37] is one of the most effective retrieval models in the language modeling framework. The relevance score is computed based on the negative KL-divergence between query model θ_Q and document model θ_D

$$S(Q, D) = -D(\theta_Q||\theta_D) = -\sum_{w \in V} p(w|\theta_Q) \log \frac{p(w|\theta_Q)}{p(w|\theta_D)}$$

where V is the set of words in our vocabulary. The document model θ_D needs to be smoothed and an effective method is Dirichlet smoothing [90]: $p(w|\theta_D) = \frac{c(w,D)+\mu p(w|\mathcal{C})}{|D|+\mu}$ where $p(w|\mathcal{C})$ is the collection language model and μ is a smoothing parameter.

Unlike vector space models, it is not natural to directly modify a query model using negative information in language model since no term can have a negative probability. In our recent work [69], several methods have been proposed for negative feedback in the language model framework. We adopt the methods there and combine them with the two heuristics discussed earlier for document penalization.

SingleNeg Strategy

SingleNeg adjusts the original relevance score of a document with a single negative model. Let θ_Q be the estimated query model for query Q and θ_D be the estimated document model for document D. Let θ_N be a negative topic model estimated based on the negative feedback documents $\mathcal{N} = \{L_1, ..., L_f\}$. In SingleNeg method, the new score of document D is computed as

$$S(Q, D) = -D(\theta_Q||\theta_D) + \beta \cdot D(\theta_N||\theta_D). \tag{5.5}$$

Note that we only penalize documents selected by either of the two heuristics.

We now discuss how to estimate negative model θ_N given a set of non-relevant documents $\mathcal{N} = \{L_1, ..., L_f\}$. We use the same estimation method as discussed in [69]. In particular, we assume that all non-relevant documents are generated from a mixture of a unigram language model θ_N (to generate non-relevant information) and a background language model (to generate common words). Thus, the log-likelihood of the sample $\mathcal{N}$ is

$$L(\mathcal{N}|\theta_N) = \sum_{D \in \mathcal{N}} \sum_{w \in D} c(w, D) \log[(1 - \lambda)p(w|\theta_N) + \lambda p(w|\mathcal{C})]$$

where λ is a mixture parameter which controls the weight of the background model and the background model is estimated with $p(w|\mathcal{C}) = \frac{c(w,\mathcal{C})}{\sum_w c(w,\mathcal{C})}$. Given a fixed λ ($\lambda = 0.9$ in our experiments), a standard EM algorithm can then be used to estimate parameters $p(w|\theta_N)$. The result of the EM algorithm gives a discriminative negative model θ_N which eliminates background noise.

SingleQuery Strategy

SingleQuery method is to update original query with negative information. Since every term has a non-negative probability in a query model, there is no natural way to update original queries with negative information. However, given Equation (5.5), a SingleQuery method can be derived after applying algebra transformation and ignoring constants that

do not affect document ranking in the following way

$$\begin{aligned} S(Q,D) &= -D(\theta_Q||\theta_D) + \beta \cdot D(\theta_N||\theta_D) \\ &\stackrel{\text{rank}}{=} \sum_{w \in V} [p(w|\theta_Q) - \beta \cdot p(w|\theta_N)] \log p(w|\theta_D) \end{aligned}$$

The above equation shows that the weight of term w is $[p(w|\theta_Q) - \beta \cdot p(w|\theta_N)] \log p(w|\theta_D)$, which penalizes a term that has high probability in the negative topic model θ_N. In this way, $[p(w|\theta_Q) - \beta \cdot p(w|\theta_N)]$ can be regarded as the updated query model, which in some sense is the language modeling version of Rocchio. For consistence with vector space model, we use γ to replace β and use $[p(w|\theta_Q) - \gamma \cdot p(w|\theta_N)]$ as the updated query model. For this query model, we use the equation above to rerank all the documents in $\mathcal{U}$. Note that for SingleQuery method, we can not apply the two penalization heuristics.

MultiNeg Strategy

MultiNeg adjusts the original relevance scores with multiple negative models. We use the same EM algorithm as SingleNeg to estimate a negative model θ_i for each individual negative document L_i in $\mathcal{N}$. We then obtain f negative models and combine them as

$$\begin{aligned} S_{combined}(Q,D) &= S(Q,D) - \beta \cdot S(Q_{neg}, D) \\ &= S(Q,D) - \beta \cdot \max(\textstyle\bigcup_{i=1}^{f} \{S(Q^i_{neg}, D)\}) \\ &= -D(\theta_Q||\theta_D) + \beta \cdot \min(\textstyle\bigcup_{i=1}^{f} \{D(\theta_i||\theta_D)\}). \end{aligned}$$

5.3 Create Test Collections with Sampling

In order to evaluate the effectiveness of negative feedback methods, it is necessary to have test collections with sufficient difficult topics. However, TREC collections do not have many naturally difficult queries. In this section, we describe two sampling strategies to construct simulated test collections by converting easy topics to difficult topics.

In our problem formulation, a query is considered to be difficult if none of the top 10 documents retrieved by a retrieval model is relevant. Thus, in order to convert an easy query to a difficult one, our main idea of sampling methods is to delete some relevant documents

of an easy query and assume these documents do not exist in the collection so that all top 10 documents are non-relevant. We now discuss two different ways to delete relevant documents:

Minimum Deletion Method: Given a query and a ranked document list for the query, we keep deleting the top ranked relevant document until none of the top 10 ranked documents of the list is relevant. We assume that the deleted relevant documents do not exist in the collection.

Random Deletion Method: Given a query and all of its relevant documents, we randomly delete a relevant document each time until none of the top 10 documents of the ranked list is relevant. Again, we assume that the deleted documents do not exist in the collection.

In both methods, we keep deleting relevant documents until none of top 10 ranked documents is relevant. Note that the constructed collections are dependent on retrieval models. After deletion, we obtain a new ranked list whose top 10 documents are irrelevant for a query. We then use these 10 irrelevant documents for negative feedback to rerank the next 1000 documents in this new ranked list.

5.4 Experiments

To evaluate the effectiveness of negative feedback techniques, we construct our test collections based on two representative TREC data sets: ROBUST track and Web track data sets.

5.4.1 Data Sets

Our first data set is from the ROBUST track of TREC 2004. It has about 528,000 news articles [67]. On average, each document has 467 terms. We use all the 249 queries as our base query set. This data set is denoted by "ROBUST."

The second data set is the GOV data set used in the Web track of TREC 2003 and 2004. It is about 18 GB in size and contains $1,247,753$ Web pages crawled from the ".gov" domain in 2002. On average, each document has 1,094 terms. In our experiment, we only

ROBUST		GOV	
LM	VSM	LM	VSM
$\mu = 2000$	$k_1 = 1.0, b = 0.3$	$\mu = 100$	$k_1 = 4.2, b = 0.8$

Table 5.1: Optimal parameter values.

use the content of the pages for retrieval. There are 3 types of queries used in Web track: homepage finding, named page finding, and topic distillation. We use the queries with topic distillation type in both Web track 2003 and 2004. In total, we have 125 queries in our base set (50 from Web track 2003 and 75 from Web track 2004). We denote this data set by "GOV."

For both data sets, preprocessing involves only stemming but without removing any stopword. Since our goal is to study difficult queries, we construct different types of query sets from our base sets as follows.

Naturally Difficult Queries

The first type of query set consists of those naturally difficult queries. In this chapter, we say that a query is a naturally difficult query if its P@10=0, given a retrieval model.

For both language models (LM) and vector space models (VSM), we use their standard ranking functions to select their naturally difficult queries respectively. We first optimize the parameters of μ for LM and k_1 and b for the VSM using the base set of queries on each data set. The optimal parameters are shown in Table 5.1. All these parameters are fixed in all the following experiments. Using the optimal parameter setting, we then select those queries whose P@10=0 as our naturally difficult queries. The row of QS0 in Table 5.2 shows the number of queries in this type of query sets.

Simulated Difficult Queries

Since there are not many naturally difficult queries, we further used the two deletion-based sampling methods to construct simulated difficult queries from easy ones. In our experiments, we use two types of easy queries. The first type consists of those queries whose P@10 satisfy $0.1 \leq$P@10≤ 0.2 (QS12 in Table 5.2) and the second consists of those

Query Sets	ROBUST		GOV	
	LM	VSM	LM	VSM
QS0: P@10=0	26	25	54	63
QS12: 0.1 $\leq$P@10$\leq$ 0.2	57	61	56	46
QS46: 0.4 $\leq$P@10$\leq$ 0.6	67	78	6	6
ALL	150	164	116	115

Table 5.2: The query sets used in our experiments.

queries whose P@10 satisfy 0.4 $\leq$P@10$\leq$ 0.6 (QS46 in Table 5.2). Again, all these queries are selected for the two retrieval models on the two data sets respectively.

The last type of query sets is the ALL query sets which are the union of the three types of query sets. Table 5.2 gives a summary of all the query sets used in our experiments.

5.4.2 Retrieval Effectiveness

Our experiment setup follows Section 5.2 to rerank the next unseen 1000 documents. We use two sets of performance measures: (1) Mean Average Precision (MAP) and Geometric Mean Average Precision (GMAP), which serve as good measures of the overall ranking accuracy. (2) Mean Reciprocal Rank (MRR) and Precision at 10 documents (P@10), which reflect the utility for users who only read the very top ranked documents.

Apply Existing Relevance Feedback

In this section, we use the Rocchio method in VSM to show that existing relevance feedback techniques do not work well if we only have negative information. The standard Rocchio method updates a query as

$$\overrightarrow{Q}_{new} = \alpha \times \overrightarrow{Q} + \beta \times \frac{1}{|\mathcal{R}|} \sum_{D \in \mathcal{R}} \overrightarrow{D} - \gamma \times \frac{1}{|\mathcal{N}|} \sum_{D \in \mathcal{N}} \overrightarrow{D}$$

where $\mathcal{R}$ is the set of relevant feedback documents. We use the query sets of "ALL" type. For any query, we first obtain its original ranking list. Starting from the top of the ranking list, we search downward until we arrive at a *cutting point*, before which we just find 10 irrelevant documents. All the documents before the cutting points, including both relevant and non-relevant documents, are used in the Rocchio feedback. The updated

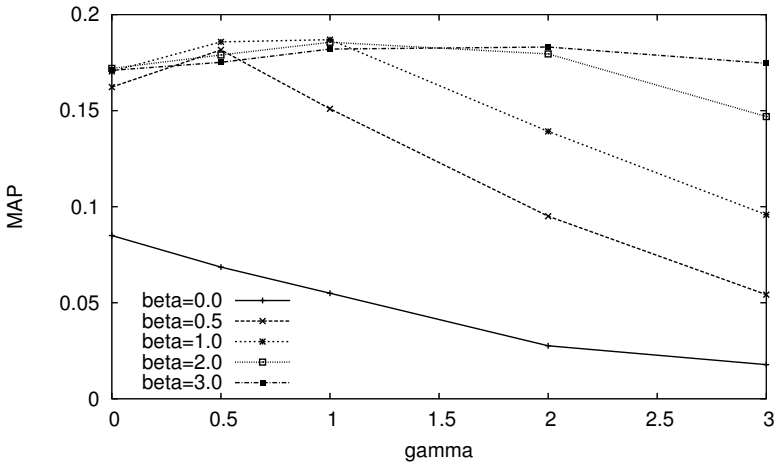

Figure 5.1: The performance of Rocchio feedback under different parameters.

query vectors are then used to rerank the next 1000 documents starting from the cutting points. In our experiments, we set $\alpha = 1.0$ and vary β and γ. The results are shown in Figure 5.1. From this figure, we can see that if we have relevant information, i.e., $\beta > 0$, the MAP values can be improved dramatically. However, when we do not have any relevant information, i.e., $\beta = 0$, negative information always hurts MAP. This means that the existing relevance feedback techniques are not effective if only negative information is available for feedback, although they are very effective for positive feedback. This also shows that special techniques are needed for negative relevance feedback.

Overall Accuracy Comparison

Using naturally difficult query sets QS0, in this section, we study the effect of different negative feedback techniques. For both LM and VSM on the two data sets, we show their performance of the original ranking (OriginalRank) and the 5 negative feedback methods: SingleQuery means the SingleQuery strategy; SingleNeg1 and SingleNeg2 are the SingleNeg strategy plus Heuristic 1 and Heuristic 2 respectively; MultiNeg1 and MultiNeg2 are the MultiNeg strategy plus Heuristic 1 and Heuristic 2 respectively.

We vary the parameters for each method: γ from 0.01 to 1 for SingleQuery, β from 0.1 to 0.9 and ρ from 50 to 1000 for SingleNeg and MultiNeg methods. In Table 5.3[1]., we

[1]Sign * and + mean improvements over OriginalRank and SingleQuery, respectively, are statistically

ROBUST+LM				
	MAP	GMAP	MRR	P@10
OriginalRank	0.0293	0.0137	0.1479	0.076
SingleQuery	0.0325	0.0141	0.2020	0.076
SingleNeg1	0.0325*	0.0147	0.2177	0.084
SingleNeg2	0.0330^{*+}	0.0149	0.2130	**0.088**
MultiNeg1	0.0346^{*+}	**0.0150**	**0.2368**	0.072
MultiNeg2	**0.0363**$^{*+}$	0.0148	0.2227	0.076
ROBUST+VSM				
	MAP	GMAP	MRR	P@10
OriginalRank	0.0223	0.0097	0.0744	**0.0416**
SingleQuery	0.0222	0.0097	0.0629	0.0375
SingleNeg1	0.0225^{*+}	0.0097	0.0749	0.0375
SingleNeg2	0.0226^{*+}	0.0097	0.0739	0.0375
MultiNeg1	0.0226^{*+}	0.0099	0.0815	0.0375
MultiNeg2	**0.0233**$^{*+}$	**0.0100**	**0.0855**	**0.0416**
GOV+LM				
	MAP	GMAP	MRR	P@10
OriginalRank	0.0257	0.0054	0.0870	**0.0277**
SingleQuery	0.0297	0.0056	0.1070	**0.0277**
SingleNeg1	0.0300*	0.0056	0.1013	**0.0277**
SingleNeg2	0.0289*	0.0055	0.0899	0.0259
MultiNeg1	**0.0331**$^{*+}$	**0.0058**	**0.1150**	0.0259
MultiNeg2	0.0311^{*+}	0.0057	0.1071	**0.0277**
GOV+VSM				
	MAP	GMAP	MRR	P@10
OriginalRank	0.0290	0.0035	0.0933	0.0206
SingleQuery	0.0301	**0.0038**	0.1085	0.0349
SingleNeg1	**0.0331***	**0.0038**	**0.1089**	**0.0396**
SingleNeg2	0.0298*	0.0036	0.0937	0.0349
MultiNeg1	0.0294	0.0036	0.0990	0.0349
MultiNeg2	0.0290	0.0036	0.0985	0.0333

Table 5.3: Optimal results of LM and VSM on the ROBUST and GOV data sets.

	POS	NEG	MNEG
Group2 relevant	0.0039	0.0032	0.0081
Group2 irrelevant	0.0024	0.0034	0.0096

Table 5.4: Similarity between POS, NEG, MNEG learned from Group1 and relevant/irrelevant documents in Group2.

compare the optimal performance (selected according to GMAP measure) of all methods. From this table, we have the following observations:

(1) LM approaches usually work better than VSM approaches. On the ROBUST data, LM can improve the MAP from 0.0293 to 0.0363, 23.8% relative improvement, but VSM can only improve from 0.0223 to 0.0233, 4.4% relative improvement. On the GOV data, LM approaches can significantly improve over both OriginalRank and SingleQuery approaches, but VSM approaches can not consistently give improvements.

(2) For LM approaches, MultiNeg always works better than SingleQuery and SingleNeg. This shows that irrelevant documents may distract in different ways and do not form a coherent cluster. To verify this, we use the cutting point defined in Section 5.4.2 to form two groups of documents for each query: all documents before the cutting point form Group1 and the next 50 documents after the cutting point form Group2. We learn a positive (denoted as POS) and a negative language model (denoted as NEG) using the relevant and non-relevant documents in Group1. Using the exponential transform of negative KL-divergence as the similarity measure, we calculate the average similarity between POS/NEG and relevant/irrelevant documents of Group2. The average values over all queries are shown in Table 5.4. We can see that POS has a notably higher similarity to relevant documents than to irrelevant documents in Group2, but NEG does not have a notably higher similarity to irrelevant than relevant documents. In this table, we also show the results of multiple negative models (denoted as MNEG). Clearly, MNEG can distinguish between relevant and irrelevant documents better than NEG, confirming that negative documents are more diverse and MultiNeg is more appropriate for negative feedback.

(3) The results of VSM are mixed, and MultiNeg can not yield notable improvement on

significant. We only show the significance tests on MAP values. Note that the values are not comparable across tables since each table corresponds to a different QS0 query set in Table 5.2

G37-07-3260432		G43-41-3966440	
VSM	LM	VSM	LM
xxxxx 22.15	mine 0.1166	pest 17.16	pest 0.1052
csic 20.85	industri 0.0475	mgmt 16.82	safeti 0.0861
quarri 20.13	miner 0.0357	4294 15.91	ds 0.0600
naic 19.48	metal 0.0319	ds 14.30	mgmt 0.0451
bitumin 18.65	or 0.0305	ipm 13.46	nih 0.0436

Table 5.5: Two examples of extracted negative models.

the GOV data. One possible reason is that the negative query vector generated using one single document in MultiNeg tends to over-emphasize rare terms due their high IDF values. In Table 5.5, we show two documents G37-07-3260432 and G43-41-3966440 in the GOV data set and their high-weight terms in the extracted negative query vectors. It is clear that VSM is biased towards those rare words such as "xxxxx" and "4294", which makes the computed negative vectors less powerful to push down those similar irrelevant documents. For LM, the extracted terms are much better. This means that LM is more powerful to pick up more meaningful terms from negative documents and thus works better on GOV data.

This may also explain why SingleNeg in VSM is generally more effective than MultiNeg on the GOV data set: SingleNeg uses *multiple* negative documents to compute the negative models. While the rare words may bias the negative model computed from *a single* negative document in MultiNeg, their weights are small in SingleNeg since they are not common in all the negative documents.

Results with Simulated Difficult Queries

We have proposed two deletion-based sampling methods to make an easy query artificially difficult. In this section, we show the retrieval results on simulated difficult queries using ALL query sets. We only show the GMAP values in Table 5.6. For Random Deletion, we run it 10 times and the average performance values are reported here. In this table, we show the results of both deletion methods on both retrieval models and both data sets. Since Random Deletion deletes more relevant documents for each query than Minimum Deletion, it is expected that its overall performance is much lower than that of Minimum Deletion.

	ROBUST+LM		ROBUST+VSM		GOV+LM		GOV+VSM	
	Minimum	Random	Minimum	Random	Minimum	Random	Minimum	Random
OriginalRank	0.0468	0.0126	0.0455	0.0115	0.0116	0.0069	0.0094	0.0056
SingleQuery	0.0467	0.0126	0.0454	0.0114	0.0119	0.0071	0.0104	0.0061
SingleNeg1	0.0473	0.0127	0.0451	0.0114	0.0118	0.0071	**0.0105**	**0.0063**
SingleNeg2	0.0475	0.0127	0.0454	0.0114	0.0120	0.0071	0.0103	0.0061
MultipleNeg1	0.0486	0.0129	0.0460	0.0116	**0.0129**	**0.0075**	0.0101	0.0059
MultipleNeg2	**0.0487**	**0.0130**	**0.0465**	**0.0117**	**0.0129**	0.0074	0.0100	0.0059

Table 5.6: The GMAP values of different methods on the simulated difficult query sets using Minimum and Random Deletion methods.

	ROBUST+LM		ROBUST+VSM		GOV+LM		GOV+VSM	
	MAP	GMAP	MAP	GMAP	MAP	GMAP	MAP	GMAP
Minimum	0.2990	0.4417	0.5015	0.7815	0.5382	0.7608	0.4932	0.6907
Random	0.3387	0.4537	0.3577	0.7417	0.5669	0.8071	0.5779	0.7271

Table 5.7: Kendall's τ coefficients between naturally difficult and simulated difficult queries.

The relative performance of different negative feedback methods is, however, similar to what we observed on the naturally difficult query sets, further confirming that the effectiveness of LM approaches and the MultiNeg strategy.

5.4.3 Effectiveness of Sampling Methods

Evaluation Methodology

To see whether a simulated test set generated using our sampling methods is as good as a test set with naturally difficult queries for evaluating negative feedback, we use both to rank different negative feedback methods based on their retrieval accuracy (e.g., MAP) and compare the two rankings; if they are highly correlated, it would indicate that the simulated test set can approximate a "natural" data set well.

Formally, assume that we have n negative retrieval functions. We can rank them based on their performance on the "gold standard" set (i.e., the naturally difficult queries). This would be our "gold standard ranking." Similarly, we can use the simulated difficult queries to rank all these retrieval functions. We then compute the correlation between these two ranking lists based on Kendall's τ rank coefficient. Given two ranking lists r_1 and r_2 of n retrieval functions, the coefficient is defined as

$$\tau(r_1, r_2) = \frac{2|\{(u,v) : r_1, r_2 \text{ agree on order of } (u,v), u \neq v\}|}{n \times (n-1)} - 1.$$

The range of the coefficient is between -1 and 1. When $\tau(r_1, r_2) > 0$, r_1 and r_2 are positively correlated. The larger the value, the higher the correlation. $\tau(r_1, r_2) = 1$ if r_1 and r_2 are exactly the same.

Results

We construct the simulated difficult queries using both Minimum and Random Deletion methods. Again we run the Random method 10 times and uses the average values to rank retrieval functions.

Our retrieval functions are from the 5 methods. For each method, we vary its parameter setting in a certain range. Each parameter setting will give us a different retrieval function. In total we have 110 retrieval functions.

Table 5.7 shows the Kendall's τ correlation coefficients between the naturally difficult queries QS0 and the simulated difficult queries on ALL query sets using the two deletion methods. From this table, we can see that both deletion methods are positively correlated with the naturally difficult queries. This confirm that our two deletion methods are reasonable to convert an easy query to a difficult one. Overall, Random Deletion is better than Minimum Deletion. Comparing two measures GMAP and MAP, we can see that the simulated difficult queries are more consistent with the naturally difficult queries on the GMAP measure. This indicates that GMAP is more appropriate as a measure on the simulated difficult queries than MAP. Indeed, GMAP has been used in ROBUST track to evaluate difficult queries and this shows the reasonableness of our deletion methods to simulate difficult queries.

5.4.4 Parameter Sensitivity Study

In this section, we study the parameter sensitivity. Due to space limit, we only show the results on ROBUST data set with the naturally difficult query set QS0; other results are similar.

Figure 5.2 shows the impact of γ on the SingleQuery method for both LM and VSM. We can see that SingleQuery can not effectively use the negative feedback information and it is

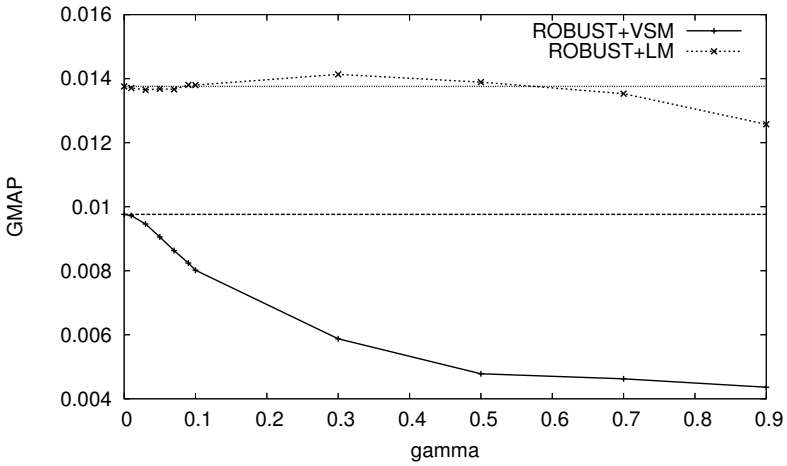

Figure 5.2: The impact of γ for SingleQuery method.

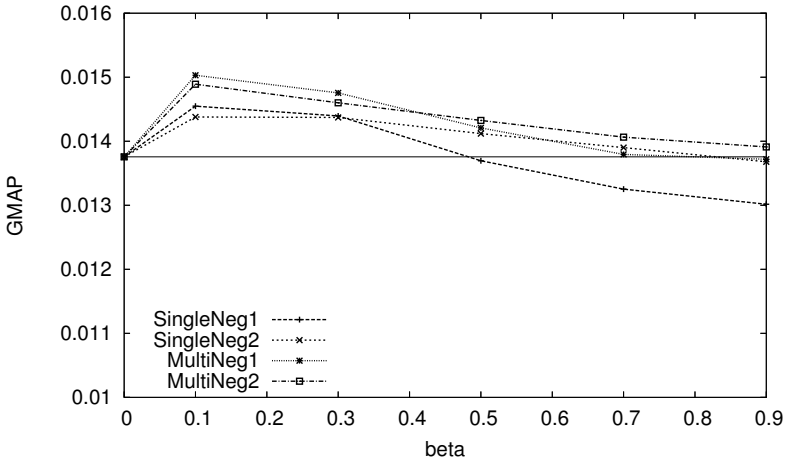

Figure 5.3: Impact of β for SingleNeg and MultiNeg.

quite sensitive if γ is larger. Figure 5.3 shows the impact of score combination parameter β where we set $\rho = 200$. All methods have the same level of sensitivities to β value. Figure 5.4 shows the impact of the penalization scope parameter ρ. It can be seen that SingleNeg1 and MultiNeg1 are very sensitive to this parameter, while SingleNeg2 and MultiNeg2 are more robust. These results confirm that Heuristic 2 is more stable than Heuristic 1 in general. Eventually, when ρ is large enough, the performance of SingleNeg2 and MultiNeg2 will drop as we penalize more documents which are not very similar to negative models. Finally, we study the impact of the number of feedback documents in MultiNeg. We set $f = 10$ but we

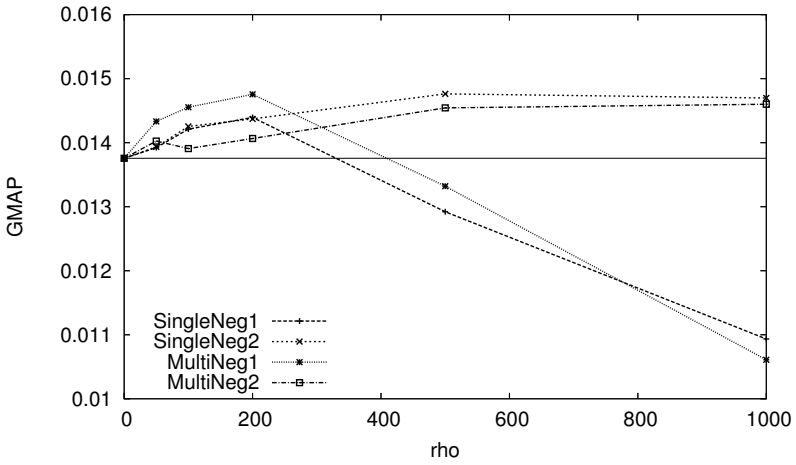

Figure 5.4: Impact of ρ in SingleNeg and MultiNeg.

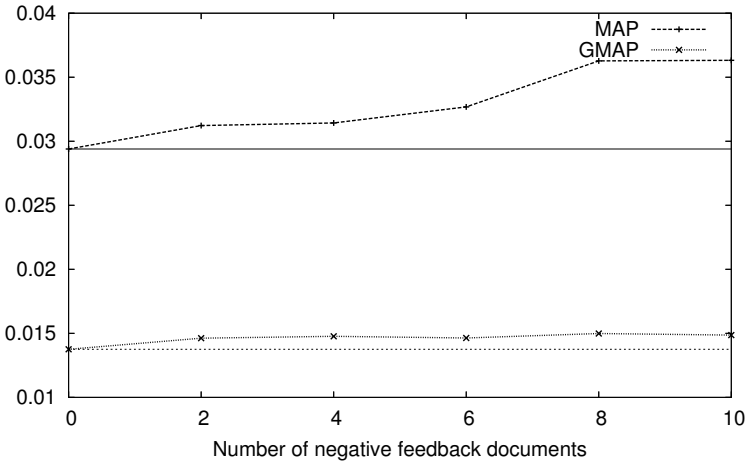

Figure 5.5: Impact of the number of feedback documents in MultiNeg.

only use a subset of these 10 documents in negative feedback. The result is in Figure 5.5 and it shows that we can get more improvement according to both MAP and GMAP if we use more documents in negative feedback. This means that our method can help more when a user accumulates more negative information.

5.5 Conclusions and Future Work

Negative feedback is very important because it can help a user when search results are very poor. In this chapter, we conducted a systematic study of negative relevance feedback

techniques. We proposed a set of general strategies for negative feedback and compared their instantiations in both vector space model and language modeling framework. We also proposed two heuristics to increase the robustness of using negative feedback information. Experiment results show that modeling multiple negative models is more effective than a single negative model and language model approaches are more effective than vector space model approaches. Studying negative feedback needs a test set with sufficient difficult queries. We further proposed two sampling methods to simulate difficult queries using easy ones. Our experiments show that both sampling methods are effective.

This work inspires several future directions. First, we can study a more principled way to model multiple negative models and use these multiple negative models to conduct constrained query expansion, for example, avoiding terms which are in negative models. Second, we are interested in a learning framework which can utilize both a little positive information (original queries) and a certain amount of negative information to learn a ranking function to help difficult queries. Third, queries are difficult due to different reasons. Identifying these reasons and customizing negative feedback strategies would be much worth studying. Forth, our study of negative feedback is constrained to a ranked list presentation. Other presentation strategies such as cluster-based presentation have been studied in the literature. An interesting direction is to study how to study negative feedback strategies with respect to other presentation strategies. For example, when none of the clusters presented to a user are relevant and the user clicks the "Next" page button, how to utilize these negative information is an interesting research topic.

Chapter 6

Support Effective Browsing I – Organizing Information Footprints

6.1 Introduction

Querying and browsing are two primary methods for users to access needed information. While querying has been proven effective for simple information needs, it alone can not satisfy complex information needs. Browsing, which is complementary to querying, can help satisfy complex information needs and discover serendipity. For difficult queries, especially exploratory information needs, it is desirable to promote effective browsing. In the following two chapters, i.e., Chapter 6 and Chapter 7, we describe our approaches to supporting effective browsing through a novel structure: multi-resolution topics maps. In this chapter, we rely on past search logs, i.e., information footprints to enhance browsing. In the next chapter, we construct a topic map based on a plain document collection.

Users' search tasks vary a lot from a simple known-item search to very complex exploratory search [78]. In a known-item search, a user has a well-defined information need and can generally formulate a very effective query and thus the current search engines often work very well. In exploratory search, however, the information need is often complex and vague, and the goal of search is mainly to gather and study information about some topic. Thus a user generally does not know well about the information to be found in exploratory search (which is the reason why the user needs to initiate the search in the first place). As a result, it is often difficult for a user to formulate effective queries in exploratory search, and the user has to reformulate queries many times in a trial-and-error manner. For example, when a user wants to buy a used car, what he/she needs is not just a single piece of information such as a list of used car dealers, but also opinions about the dealers by previous customers, advantages/disadvantages of different brands, and advice on car insurance, etc.

Formulating effective queries to find all this information is quite challenging, especially for a user who does not know well about the domain. For these reasons, the current search engines generally do not perform well for exploratory search compared with known-item search [41]. Since exploratory search happens very often, it is very important to study how to help users to conduct effective exploratory search [77, 41, 78].

Querying alone is often insufficient to support exploratory search well due to the difficulty in formulating good queries. When a user is unable to formulate effective queries, browsing would be intuitively very useful because it enables a user to navigate into relevant information (and explore the information space in general) without formulating a query. Indeed, being able to browse the Web through hyperlinks is essential to web users, and quite often, a user would find relevant information by following hyperlinks in the result pages. Had all the hyperlinks been broken, the utility of a search engine would be significantly reduced.

Unfortunately, with the current search engines, browsing is mostly through following static hyperlinks. This is very restrictive and would not allow a user to go very far in the information space. A main research question we want to study in this chapter is how to more effectively support browsing for ad hoc exploratory queries so that users can go beyond hyperlinks to freely navigate into remotely related topics in the entire information space.

There have been some efforts on providing more powerful navigation support, but they tend to rely on manually created meta data and usually can only support "vertical" navigation through hierarchies. Thus they are inadequate for supporting browsing for ad hoc queries. For example, Web directories such as Yahoo! directory[1] and Open Directory Project[2] use manually constructed hierarchies to support drill-down and roll-up. Faceted hierarchies [28, 83] go beyond a single hierarchy to support browsing with multiple hierarchies. The multiple hierarchies are carefully designed and built along different dimensions in a given domain (e.g., time or location dimensions for news articles) so that a user can flexibly choose different dimensions to narrow down their search. However, these hierar-

[1] http://dir.yahoo.com/
[2] http://www.dmoz.org/

chies are created manually and thus need a lot of human efforts to adapt them in a new domain. More importantly, they only allow users to move *vertically* (i.e., drill-down or roll-up). These two operators are not sufficient enough for users to exploit related information. When a query is difficult, horizontal navigation to neighbor topics is especially beneficial.

Thus it remains a significant challenge how to effectively support effective browsing beyond hyperlinks for arbitrary *ad-hoc queries*. Ideally we should support both vertical navigation and horizontal navigation. In this chapter, we propose to achieve this goal through a novel navigation structure: multi-resolution topic maps. A topic map is an analogy of geographical map in an information space. Technically, a topic map is an extension of hierarchy but it has two distinct features: (1) Topics in the same level of the map have similar resolution. (2) There are horizontal links between topics in the same level, besides the vertical links as in a hierarchy. In a multi-resolution topic map, topics with coarse granularities (i.e., low resolutions) subsume those with finer granularities (i.e., high resolutions). For example, "car" can subsume "car rental," "car pricing" and "car insurance." Related topics in the same granularity are connected horizontally. For example, "flight" is connected with "hotel," "vacation" and "car." With a multi-resolution topic map, a user can easily reach topics of different granularities through *vertical navigation* (i.e., zoom in and zoom out) as well as topics with the same level of granularity through *horizontal navigation* (i.e., moving to a neighbor area), achieving flexible navigation. Just as a geographical map can tour a tourist in a city, a topic map can guide a user in an information space.

To construct a multi-resolution topic map, we rely on past search engine logs, which can be regarded as "footprints" left by previous users in the information space. Just as the footprints of previous visitors can help guide future visitors to in a park, the footprints in an information space left by previous users can also help future information seekers. Our multi-resolution topic map is to capture these footprints in a semantical way so that it can guide a user to reach relevant information by following the "wisdom of crowd." Compared with past work on exploiting search logs, turning the entire search logs into a topic map is a novel way to leverage search logs to help search. Moreover, as new users use the topic map to navigate the information space, they will add footprints to the information space, which

can then be used to improve and refine the map dynamically for the benefit of future users, thus enabling users to surf the information space in a collaborative manner. The topic map essentially serves as a sustainable and continuously growing infrastructure for collaborative surfing.

The constructed topic map can be used to support flexible browsing in a topic space, which can be integrated with regular querying in a search system to enable users to flexibly query, browse a map node, and navigate over the topic map to find relevant information. We evaluate the potential benefit of our topic map using a sample of search logs from a commercial search engine. The experimental results show that the idea of supporting flexible browsing with a multi-resolution topic map is promising and our method of constructing topic maps based on search logs is effective in helping a user reach useful pages quickly through pure browsing.

The main contributions of this chapter are:

- We propose a novel structure (i.e., multi-resolution topic map) to enable a user to go beyond hyperlink following to flexibly navigate in the information space.
- We propose a novel way to exploit search logs for improving search by treating search logs as information footprints and organizing all the search logs into a topic map, which can potentially provide sustainable collaborative surfing in information space.
- We propose a general method for turning any search logs into a multi-resolution topic map based on the star-clustering algorithm.
- We evaluate the effectiveness of the novel topic map in supporting browsing and show promising results.

6.2 Multi-Resolution Topic Map

In this section, we formally define multi-resolution topic maps. Suppose we have an information space consisting of a collection of documents C. We first define *Topic Region* and *Topic Region Space*:

Definition 8 (Topic Region) *A topic region $T \subset C$ is a subset of documents that are about a topic. For example, all the documents matching a phrase can form a topic region characterized by the phrase.*

Definition 9 (Topic Region Space) *The topic region space S is the set of all possible topic regions defined on C. That is, $S = 2^C$.*

Note that for generality, we allow the topic region space to contain potentially non-coherent topic regions. We now define *Topic Map*.

Definition 10 (Topic Map) *A topic map $M = (V, E)$ is a graph with regions as vertices (i.e., $V \subset S$). An edge between two topic regions means that the user can navigate from one topic region into the other. That is, if $(v_i, v_j) \in E$, then a user would be able to navigate between v_i and v_j.*

A topic map is to guide a user navigating in the information space just as a geographic map can guide a traveller touring a city. As a geographic map would show roads to connect different regions to enable transportation, our topic map would also have semantic connections between topic regions to enable browsing.

In any interesting application, especially an unrestricted domain such as Web, the topic map can be quite large. How to facilitate a user in navigating on this map would be itself a challenge. We solve this problem by constructing a topic map with multiple resolutions. The idea of multiple resolutions is again analogous to the idea of displaying a geographic map in multiple resolutions, and it would allow a user to get to one region from another easily on the map. Specifically, if the user wants to visit a topic region far away on the map, he/she can simply "zoom out" to a high-level general topic region (e.g., sports) and quickly navigate into a quite different (general) topic region (e.g., economy); similarly, if the user is interested in a region and wants to explore more in the region, he/she can "zoom in" and get a detailed view of the region (e.g., from "sports" to a set of regions such as "baseball," "basketball," and "football").

We now define the multi-resolution topic map formally.

Definition 11 (Multi-Resolution Topic Map) *A k-level multi-resolution topic map consists of k ordered topic maps, $M = (M_1, ..., M_k)$, such that for any two adjacent maps $M_i = (V_i, E_i)$ and $M_{i+1} = (V_{i+1}, E_{i+1})$, we have a zooming relation $Z \subset V_i \times V_{i+1}$ which covers every topic region in both M_i and M_{i+1}.*

In a multi-resolution topic map, we can refine browsing into *vertical browsing* and *horizontal browsing*. The zooming relation tells us how to refocus on a map with a new resolution if the user zooms in/out on a current map. Specifically, suppose the user is currently visiting region v_i on map M_i. If the user zooms in, he/she will see a set of "children" topic regions $\{v_{i-1}|(v_{i-1}, v_i) \in Z\}$ on map M_{i-1}. Similarly, if the user zooms out, he/she will see a set of "parent" topic regions $\{v_{i+1}|(v_i, v_{i+1}) \in Z\}$ on map M_{i+1}. Thus, with the zooming relation and maps of multiple resolutions, a user can potentially navigate into remotely related topics quickly.

6.3 Search Log Based Topic Map

While a multi-resolution topic map can be constructed in many ways, in this chapter, we focus on studying how to construct such a map based on search logs. Turning search logs into a topic map to support browsing has two attractive benefits: First, since queries and clickthroughs in search logs can both be regarded as "information footprints" left by previous users in the information space, thus constructing such a map would enable the current users to follow these footprints and leverage the "wisdom of crowds." Second, as new users use the map to navigate and leave more footprints, we will be able to use the new footprints to dynamically update and refine the topic map for the benefit of future users, thus achieving a powerful naturally sustainable model of collaborative surfing.

We now present a general method for constructing a topic map to organize the information footprints in search logs. Our approach is based on an extension of the star clustering algorithm [3], which has a parameter to naturally control the granularity of the obtained topic regions, thus helps attain the goal of multiple resolutions.

6.3.1 Representing Footprints

We use both queries and clickthroughs to represent information footprints. Our method is to generate a pseudo-document for each query. We utilize the clickthrough information in search logs for this purpose. For each query in the logs, we have all the clicked URLs by all past users. However, only URL information would not give meaningful representations since URLs alone are not informative enough to capture the footprints accurately. To gather rich information, we enrich each URL with additional text content. Specifically, given any query, we can obtain its top-ranked results using the same search engine as the one from which we obtained our log data, and extract the search engine snippets of the clicked results, according to the log data. Given a query, all the snippets of its clicked URLs are used to generate a pseudo-document. Thus, each pseudo-document corresponds to a unique query and the keywords contained in the query itself can be regarded as a brief summary of the corresponding pseudo-documents. Intuitively, all these pseudo-documents and their associated queries capture the footprints in the information space and we use them to build our topic regions through clustering techniques.

6.3.2 Forming Topic Regions

Let $Q = \{q_1, ..., q_n\}$ be all the queries in the search logs and $L_0 = \{d_1, ..., d_n\}$ their corresponding pseudo-documents. We use the star clustering algorithm [3] to discover coherent topic regions.

Given L_0, star clustering starts with constructing a pairwise similarity graph on this collection based on the vector space model in information retrieval [56]. Then the clusters are formed by dense subgraphs that are star-shaped. These clusters form a cover of the similarity graph. Formally, for each of the n pseudo-documents $\{d_1, ..., d_n\}$ in the collection L_0, we compute a TF-IDF vector. Then, for each pair of documents d_i and d_j $(i \neq j)$, their similarity is computed as the cosine score of their corresponding vectors. A similarity graph G_σ can then be constructed using a similarity threshold parameter σ as follows. Each document d_i is a vertex of G_σ. If $sim(d_i, d_j) > \sigma$, there would be an edge connecting the corresponding two vertices. After the similarity graph G_σ is built, the star clustering

algorithm clusters the documents using a greedy algorithm. We outline the star clustering algorithm in Algorithm 1.

Algorithm 1 Star clustering algorithm

1: Given a parameter $\sigma(0 \leq \sigma \leq 1)$, generate a similarity graph $G_\sigma = (V, E)$.
2: Associate a flag $I(v) = unmarked$ for $\forall v \in V$.
3: **repeat**
4: Let $u = \arg\max_{I(v)=unmarked} degree(v)$, i.e., u is the *unmarked* vertex with the largest degree.
5: Mark $I(u) = center$.
6: Form a cluster $C_u : \{u\} \bigcup \{v : (u, v) \in E\}$ where u is the center of the cluster.
7: Mark $I(v) = satellite$ if $(u, v) \in E$.
8: **until** $I(v) \neq unmarked$ for $\forall v \in V$.

In star clustering, each obtained cluster is *star-shaped*, which consists a single *center* and several *satellites*. There is only one parameter σ in the star clustering algorithm. A big σ enforces that the connected documents have high similarities, and thus the clusters tend to be small. Such a small cluster corresponds a topic region with finer granularity. On the other hand, a small σ will make the clusters big and such a cluster corresponds a topic region with coarse granularity.

6.3.3 Building a Multi-Resolution Topic Map

For a multi-resolution topic map, we can build it in either a top-down or a bottom-up manner. In this section, we adopt a bottom-up hierarchical clustering method.

Generating Multi-Resolution Map Nodes

We use hierarchical star clustering to build map nodes and their zooming relations. Let L_0 be the set of individual queries. We apply our star clustering algorithm on L_0 with a high σ_1 values so that we can find small but very coherent topic regions. Each region/cluster provides a *center* query and all these center queries form a set L_1. Recursively, we can apply star clustering on L_1 with a medium threshold σ_2 to generate another set of center queries L_2. L_2 can then be used to generate L_3 with a small threshold σ_3 and etc. In our experiments, we generate a three-level topic map by setting $\sigma_1 = 0.7$, $\sigma_2 = 0.5$, and $\sigma_3 = 0.3$. Recursive clustering gives us clusters in different granularities. Since we have the

same threshold σ for each level, we loosely ensure that all the topics in the same level have similar granularities. Each cluster is a node in our map and all clusters in L_i form the set of nodes in i-th level of our map.

Connecting Topic Regions for Browsing

The procedure above generates a k-level hierarchy which can support vertical zoom in/out naturally: A cluster in a coarse granularity subsumes several clusters in a finer granularity. Thus in our map, we have vertical or zooming relations among the corresponding nodes. Each cluster in different levels is a topic region which contains a set of pseudo-documents in L_0 and a set of queries.

Here we describe our methods to connect nodes/clusters in the same level to support horizontal navigation. In the same level, each cluster has a set of queries in Q and all these queries in the set can be used as the content of the cluster. Intuitively, semantically closely related clusters would have high similarities in their contents. Therefore, we can build a vector representation for each cluster and use cosine similarity score to measure the closeness of two clusters. In this chapter, we propose a random walk based similarity measure which can be used to incorporate other useful information in logs such as query sequences in user sessions.

Specifically, given two clusters C_i and C_j, we would calculate a probability $P(C_j|C_i)$ to measure the probability of arriving at cluster C_j if we start a random walk from C_i. The general random walk works as follows: From C_i, we randomly walk to a query $Q_b \in C_i$. Then we randomly walk to another query Q_a from Q_b. The last step is another random walk from Q_a to a cluster C_j which contains Q_a. Therefore

$$P(C_j|C_i) = \sum_{Q_a,Q_b} P(C_j|Q_a)P(Q_a|Q_b)P(Q_b|C_i). \tag{6.1}$$

All those probabilities can be modelled flexibly. For example, $P(Q_a|Q_b)$ can be modelled as the probability of a user reformulates queries from Q_b to Q_a. Another version of random walk is to change Q_a and Q_b to two terms w_a and w_b respectively. Then we have a similar

formula

$$P(C_j|C_i) = \sum_{w_a,w_b} P(C_j|w_a)P(w_a|w_b)P(w_b|C_i). \tag{6.2}$$

where $P(w_a|w_b)$ can be modelled as the probability of seeing w_a in a following query given its previous query containing w_b in user sessions. Without using any additional information, we can assume $P(w_a|w_b) = 1$ if $w_a = w_b$ and 0 otherwise. Then Equation (6.2) can be simplified as

$$P(C_j|C_i) = \sum_{w} P(C_j|w)P(w|C_i). \tag{6.3}$$

In our experiments, we use Equation 6.3 and estimation $P(w|C_i) = \frac{c(w,C_i)}{\sum_w c(w,C_i)}$ and $P(C_j|w) = \frac{c(w,C_j)}{\sum_C c(w,C)}$ where $c(w, C)$ is the count of w appearing as a content word in cluster C.

Labelling Map Nodes

Each cluster generated above corresponds to a node in our topic map. To provide effective guidance when end users navigate in our topic map, we need to associate a meaningful label with each node. A label should be informative enough to represent the node's content in the corresponding cluster. Similar to [72], We use query words to generate labels for each node in our map since query words are more accessible from a user's viewpoint. In this chapter, we use a variant of frequent pattern algorithm to generate the labels in a top-down manner. We start from the nodes in the highest level (Level 3) of our map. For each node, we take every query in the corresponding cluster as a word sequence and find the most frequent one (unigram) or two words (bigram) in the corresponding query set as its label. For example, we can get a label "car" for a node in Level 3. After generating labels for Level 3, we apply the similar procedure to Level 2, but with a constraint that a word will not be selected if it has been used by its parent node. After we get the frequent word(s) for a node in Level 2, we *append* the label of the node's parent node in Level 3 as prefix to label the node. For example, if we get the most frequent word of a node in Level 2 as "rental" and the node's parent's label is "car", then we label the node by "car::rental". For a node in Level 1, we use the *center* queries output by the star clustering algorithm as its label.

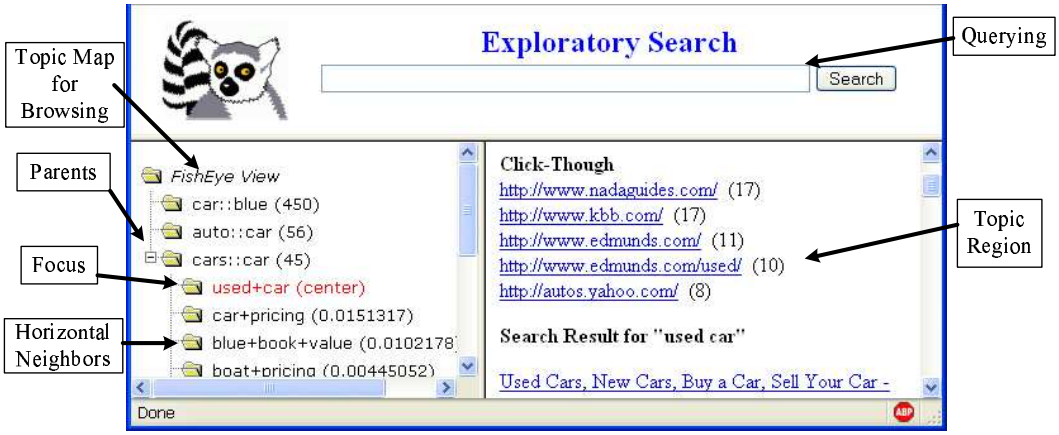

Figure 6.1: Interface snapshot of our topic map-based browsing.

6.4 Browsing with Topic Maps

Once a topic map is built, we can integrate it into a regular search engine to enhance browsing. We developed a prototype system based on a map constructed using a sample of search logs and a commercial search engine. A snapshot of our system interface is shown in Figure 6.1. In our system, a user has access to three operators all the time: querying, viewing a map node, and navigating in the map.

Querying. When a user submits a new query through the search box, the search results from a search engine will be shown in the right pane. At the same time, we build a "query-extended" map by connecting the query defined topic region with its closest map nodes in Level 1. The closeness is computed as follows: given the query, we first retrieve the top m pseudo-documents using the standard Okapi method [51]. Each pseudo-document corresponds to a past query. For nodes/clusters in Level 1, we count how many of the retrieved pseudo-documents each contains and use these counts as the closeness measure. The closest map nodes are then ordered accordingly and shown in the left pane of Figure 6.1.

Viewing a map node. When a user *double* clicks on a map node, we are going to display the topic region corresponding to the current node on the right pane. In this chapter, the topic region consists of two parts: (1) the clicked URLs of all the past queries in the current map node, and (2) the returned search results of using the label of the current map node as a query. The content in the right pane tells a user the most frequently visited pages for in

the current node and also the search results. The user can thus visit footprints of previous users or leave his/her own footprints by examining new search results.

Navigating in the map. The left pane in our interface is to let a user navigate in the map. When a user clicks on a map node, this pane will be refreshed and a fisheye view with the clicked node as the current focus will be displayed. In a fisheye view, we show the parents, the children, and the horizontal neighbors of the current node in focus (labelled as "center" in our interface). A user can thus zoom into a child node, zoom out to a parent node, or navigate into a horizontal neighbor node. In our current implementation, the children and neighbor nodes are ordered by Equation 6.3 and the parent nodes are ordered by their size (the number of children they contain).

The three different operators provide flexibility for users to either querying or browsing interchangeably. Navigating in the map provide semantic roads to help user reach related topic regions even without formulating a query by himself/herself.

6.5 Experiments

6.5.1 Data Set

Our data set is a sample of search log data from a commercial search engine. In total, this log data spans 31 days from 05/01/2006 to 05/31/2006; there are 8,144,000 queries, 3,441,000 distinct queries, and 4,649,000 distinct URLs in the raw data.

To test our system, we separate the whole data set into two parts according to the time: the first 2/3 data is used to simulate the historical data that a search engine accumulated. We treat this log data as footprints and build our topic map. The last 1/3 data is held out to serve as our test cases, which will be described in detail in a later section. In the history collection, we clean the logs by only keeping those frequent, well-formatted, English queries (queries which only contain characters 'a', 'b', ..., 'z', and space, and appear more than 5 times). After cleaning, we get 169,057 unique queries in our history collection in total. On average, each query has 3.5 distinct clicks. For each query, we build a "pseudo-document" based on its clicked snippets. The average length of these pseudo-documents is 68 words

and the total data size of our history collection is 129MB.

6.5.2 Three-Level Topic Map

Based on the history collection we described above, we built a three-level topic map according to the method we described in Section 6.3. The first level has the finest granularity and the third level has the most coarse granularity. We show a part of the nodes/clusters in different levels in Figure 6.2 where each node is represented by its label words and arrows and lines denote vertical and horizontal relations, respectively. We use past queries as labels for the first-level nodes, but we use query words instead of entire queries to label the nodes/clusters in the second and third levels. A major reason for doing this is that user queries tend to be very specific, so they may not always be suitable to label clusters in a coarse granularity, and using query words may be better. From this figure, we can see that the nodes/clusters in the first level are relatively narrow topics such as "alamo car rental." On the other hand, the second and third level clusters represent more general concepts such as "car." On the same level, we have horizontal neighbors whose closeness is calculated by random walk based similarity in Equation 6.3. We can see that the closeset neighbors are indeed related. For example, we can go from "car" to "auto," to "loan," or to "insurance." All these neighbors provide useful guidance/choices for users to navigate into related topic regions.

6.5.3 Effectiveness of Map-based Browsing

Experiment Design

To test the effectiveness of our idea of using search logs as footprints, we construct our test cases using the sessions in our held-out test logs. The test logs consist of user sessions and in each session, a user submitted several queries sequentially and clicked certain documents for each of the submitted queries. A typical scenario is that a user first tries an initial query and clicked on certain documents. If the current results are poor or the user wants to find more relevant information, the user would reformulate queries several times and click on more documents. In our experiments, we use each session as a test case and use the clicked

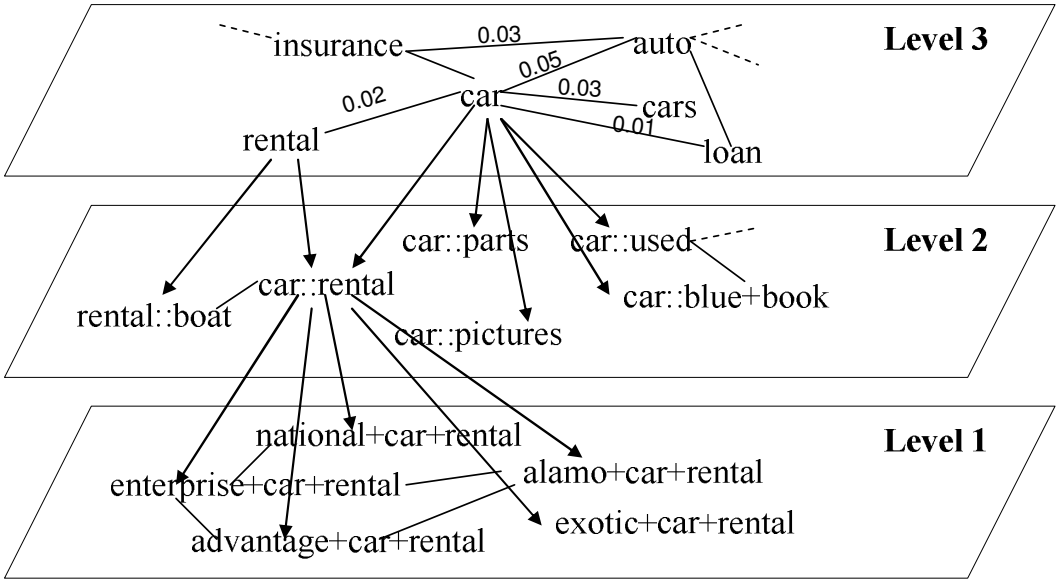

Figure 6.2: Examples of map nodes in the three-level topic map.

documents in a session to *approximate* the relevant documents [33]. In a search session recorded in the search logs, the user tried to find additional relevant documents through repeatedly reformulating queries. Our idea is to do a simulation evaluation and see whether a user would have been able to find such additional relevant documents more effectively only through browsing the topic map.

Formally, let $\{Q_1, Q_2, ..., Q_k\}$ be a sequence of queries that a user tried in a session and let R be all the clicked documents for queries $\{Q_2, ..., Q_k\}$. R is regarded as the *additional* relevant documents to the user's information need. Note that we do not include the clicked documents of Q_1 in R because we want to simulate the scenario of a user browsing a map to find more relevant documents *after* finishing the search with the first query instead of formulating additional queries to find more relevant documents (the clicked documents of Q_1 would presumably have already been seen by the user by this point). Our experiments are designed to test how effective our guided navigation is in enabling a user to reach the documents in R *only* through browsing.

As expected, most sessions in our logs are not exploratory search. Since our goal is to support exploratory search, we use several heuristics to filter those sessions in our test cases. Each session has at least 2 different queries and at least 10 clicked documents (including the

clicks for Q_1). To ensure that queries in a session are about a coherent information need, we further require that two adjacent queries in a session should share at least one word. After applying the above heuristics to our test data, we obtain 76 sessions as our test cases. On average, each session has 2.22 queries and the size of R is 7.74.

To evaluate our methods, we conduct experiments to simulate a user's actions when the user uses our system. In particular, we simulate a one-step action which is to simulate that a user views 1 node in our map after the user submits the very first query Q_1. We will compare the benefit of this navigation action in our system with a query reformulation action of submitting a second query Q_2.

We compare our methods with two baselines. Our first baseline method (BL1) is to use Q_1 to retrieve a ranked list from a search engine. Our second baseline (BL2) is to use Q_2 to retrieve documents from the same search engine. We use R to evaluate the accuracy of these two baselines. For our method, we use Q_1 as input to return a list of map nodes in Level 1 to a user. Then the user can first *examine* several nodes and finally decide to *view* a returned node. After the user views a node in our map, a ranked list of URLs of previously clicked documents in the node/cluster will be presented, as well as a list of organic search results from the search engine (refer to Section 6.4). For simplicity, we rank all the clicked URLs on the top of the search results and their rankings are decided by the historical click frequencies (see Figure 6.1). We then use R as relevant set to evaluate the returned URL lists after the user views a map node. To simulate which node a user will view in our map, we use 4 variants as follows.

Simu0Default: This variant is the most naive method which assumes that the user will view 1st ranked map node.

Simu0Best: This variant is to assume that the user will view the "best node" after examining the top 10 map nodes returned for Q_1. We will describe what is the best node soon.

Simu1Default: This variant is an extension of Simu0Default. In this variant, a user single-clicks on the 1st ranked node and our system will display a fisheye view of the current node. The user then examines both the 1st ranked node and its top 10 horizontal neighbor

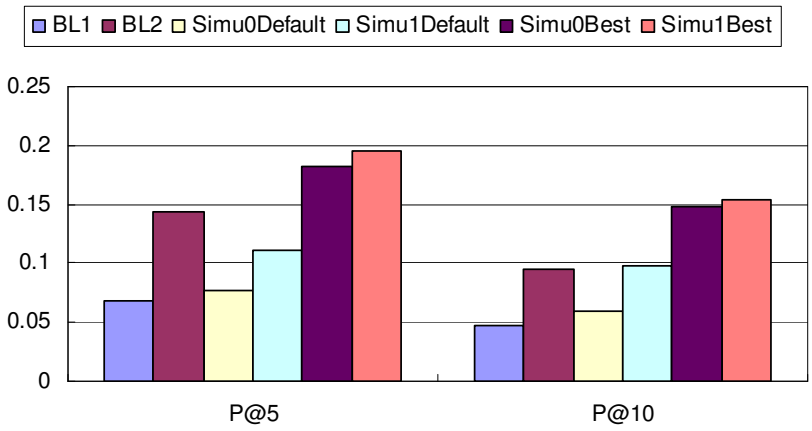

Figure 6.3: Comparison of different methods

nodes. The best node of these 11 nodes is finally *viewed* by the user.

Simu1Best: This variant is an extension of Simu0Best. In this variant, a user single-clicks on the node selected in Simu0Best and our system will display a fisheye view of the current node. The user then examine both this clicked node and its top 10 horizontal neighbors. Finally the user decides to *view* the best node among all these 11 nodes.

For all the 4 variants, Simu0Best, Simu1Default, and Simu1Best assume a user would optimally choose the best node to view, where the best node is the one whose ranked list of URLs (in its defined topic region) have the best P@10, evaluated based on R. These are optimal simulations which are to show the performance upper-bound of our system. However, given informative and accessible labels in our map, users can probably choose the best or nearly best node to view in reality. Simu1Default and Simu1Best are extensions of Simu0Default and Simu0Best, and are to test whether a user can get even more useful information after more exploration.

Treating R as the relevant documents, we use P@5 (Precision at 5 documents) and P@10 (Precision at 10 documents) to evaluate different methods.

Result Comparison and Analysis

In Figure 6.3, we compare different methods using the two primary measures. We compare the two baseline methods BL1 and BL2 with four variants of our method. In this figure, we can see that BL1 is very poor and it means that the first query is ineffective to retrieve

	#Sessions	BL1	Simu0Best	Impr.
Part I (P@10 = 0)	50	0	0.114	0.114
Part II (P@10 > 0)	26	0.138	0.173	0.035

Table 6.1: Improvement over difficult queries with respect to average P@10. Part I corresponds to those more difficult queries.

additional documents. Simu0Default is a naive method which assumes the user would view the first node. Since the first node is the most similar to the current query Q_1, it is not surprising that its result is also poor. BL2 uses the second query Q_2 and the result is much better. Intuitively, this means that reformulating queries can get more clicked documents. Comparing with BL2 based on P@10, our variant Simu0Best achieves a relative improvement of 57% and Simu1Best achieves a relative improvement of 63%. Both improvements are statistically significant according to Wilcoxon test: p-values are 0.003 and 0.002 respectively. This means that selectively viewing a node in our map can reach more relevant documents accurately than reformulating a query. This is because our map nodes aggregate clicked documents for a set of queries and viewing a node will reach a topic region which ranks the documents in a collaborative manner.

From this figure, we can also see that Simu1Default and Simu1Best achieve better accuracy than Simu0Default and Simu0Best respectively. The Wilcoxon tests show that the improvements are also significant: p-values are 0.01 and 0.02 respectively. This means that more relevant documents can be reached through *navigating* to and *viewing* a neighbor node. All these confirm the benefit of browsing and the effectiveness of topic map based approaches.

Difficult Query Analysis. We show the effectiveness of our method for difficult queries. In this experiment, we use BL1 to assess the difficulty of queries. For all the test cases, we separate them into two parts according to their P@10 in BL1. The first part (Part I) corresponds to the cases with P@10 = 0, which means Q_1 can not retrieve any additional documents to top 10. The second part (Part II) corresponds to the cases with P@10 > 0. This means that we can retrieve at least 1 document using the original query Q_1. We compare the improvement of our Simu0Best over BL1 for these two sets of test cases using P@10. Table 6.1 summarizes the results. In this table, we can see that 50 test cases fall

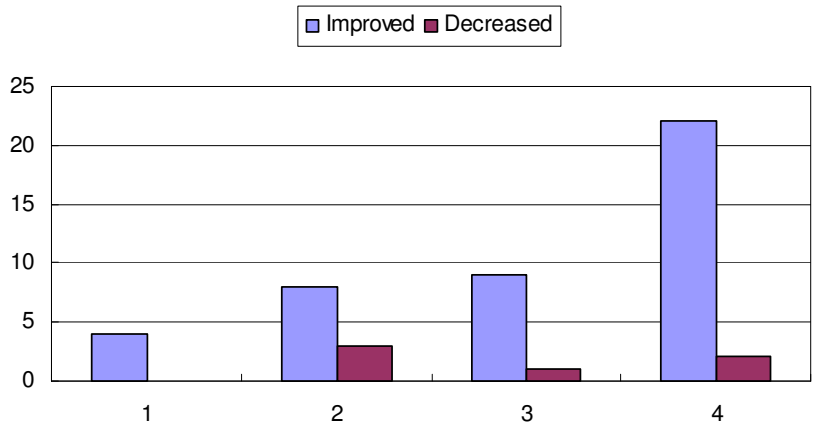

Figure 6.4: The impact of history richness.

into Part I and 26 test cases fall into Part II. For Part I, we can improve P@10 by 0.114 from 0 to 0.114 on average. For Part II, the improvement is only 0.035 from 0.138 to 0.173 on average. Since the cases in Part I is more difficult than the cases in Part II, this means that navigation based on our topic map can help more for more difficult queries.

History Richness. Our topic map is based on search logs. Different test cases have different amount of similar history information in our logs. Our hypothesis is that a test case with richer history information in our logs will benefit more from our topic map. To verify this, we use Q_2 to retrieve our history collection and use the number of returned pseudo-documents as the indicator of the history richness for the test case. According to the number of returned pseudo-documents, we separate the test cases into 4 bins. Bin 1 has 0∼40 , Bin 2 has 40∼80, Bin 3 has 80∼120, and Bin 4 has more than 120 returned pseudo-documents. Bin 1 corresponds to those cases without much history while Bin 4 corresponds to those cases having rich history. For each bin, we show the number of test cases whose P@10's are improved versus decreased, by comparing Simu0Best with BL1. The result is shown in Figure 6.4. From this figure, we can see that the percentage of improved test cases increases along with the increase of the history richness. For example, in Bin 4, we improve 22 and decrease 2 cases. But in Bin 2, we increase 8 and decrease 3. This confirms that the more history we have, the better we can help users for browsing. More importantly, as time goes, more and more queries will have sufficient history, so we can improve more and more exploratory searches, resulting in a sustainable model for effective collaborative surfing.

6.6 Discussions

A main point made in this chapter is that the current search engines would be more powerful, especially for supporting exploratory search and helping users find hard-to-find information, if they can offer better support for browsing through a multi-resolution topic map, such as the one constructed based on search logs. Although our topic map is built upon a relatively small sample of search logs, our experiment results have clearly demonstrated the feasibility of supporting browsing for ad-hoc queries in a general way. We also demonstrated that queries with more history information can benefit more from topic map-based browsing. This is very encouraging since there are more search logs in the commercial search engines that can be leveraged, and as a system is being used more, more search logs would be naturally accumulated.

Our work raises some interesting new possibilities in advancing the search engine technologies, which we will briefly discuss in this section.

6.6.1 Multi-Faceted Browsing for Ad Hoc Queries

Although we only experimented with a topic map built based on search logs in this chapter, one can easily imagine that we can also build a topic map in many other ways based on various data sources. Indeed, even with search logs, we have multiple ways to build a topic map. For example, instead of using semantic similarity of queries to construct a topic map as we have done in our experiments, we can also leverage the co-occurrence relation of queries in a user session to build an alternative topic map where related queries to the same task may be connected together (e.g., queries about "flight ticket" may be connected with those about "car rental" even though the two sets of queries may not be similar by content). Yet another way to construct a multi-resolution topic map is based on query word editing patterns [73]. In such a map, a node corresponds to a query. All queries with the same number of keywords belong to the same level. The children of a map node is obtained by adding a keyword into the current query and the neighbors of the query is by substituting a keyword in the current query.

A topic map can also be constructed by other data sources. For example, any domain-

specific ontology can also be extended to a topic map by adding horizontal relations. We can also build a topic map solely based on a plain document collection itself. We will explore this direction in the next Chapter.

With multiple topics constructed using different criteria, a search engine system would be able to potentially support multi-faceted browsing for ad hoc topics, i.e., a user would be allowed to switch between from one facet (map) to another to explore information in an extremely flexible and powerful way.

6.6.2 Unify Querying and Browsing

Querying and browsing are the two most important information seeking strategies. They are complementary, and both are needed in a search task [85]. An important research question is thus: Can we integrate querying and browsing in a *unified formal framework*? Interestingly, as we will further discuss below, it is possible to view querying as a special way of navigating in the information space, thus we can integrate querying and browsing within a single unified navigation framework.

A Formal Navigation Framework. In Section 6.2, we have defined *topic regions* and *topic region space*. Under these definitions, we can view querying as navigation in this space. More specifically, after querying, a user would end up viewing a subset of documents (i.e., search results), thus we can view this process as helping the user navigate into the topic region consisting of the search results. When a user repeatedly submits a query, the user would be essentially visiting different topic regions defined by the queries.

When a user follows a path on a topic map, the user would also be moving from one topic region to another, just like submitting reformulated queries. Thus both querying and browsing can be formalized as navigation operators defined below:

Definition 12 (Navigation Operator) *A navigation operator is a function that maps one topic region to another. We use N as the set of all navigation operators. That is, $N = \{f : S \rightarrow S\}$, where S is topic region space.*

Definition 13 (Query Navigation Operator) *A query navigation operator $Q(q)$ is defined as $Q(q)(T) = T_q$, where q is a query and T_q is the topic region corresponding to*

the search results of using the query q. For any $T_1 \neq T_2$, we have $Q(q)(T_1) = Q(q)(T_2)$. Therefore, such a definition assumes that a query navigation operator returns a topic region regardless the current region. It is thus a "memoryless" navigator and we can use $Q(q)$ to represent T_q without incurring confusion.

Definition 14 (Browsing Navigation Operator) *A browsing navigation operator $B(v_1, v_2)$ is defined as $B(v_1, v_2)(v_1) = v_2$, where $(v_1, v_2) \in E$ is an edge on the topic map. $B(v_1, v_2)(v)$ is undefined if $v \neq v_1$. Intuitively, a browsing navigation operator $B(v_1, v_2)$ brings a user from topic region v_1 to v_2.*

Definition 15 (Compatibility) *Two navigation operators N_i and N_j are compatible if and only if one of the following three conditions holds: (1) N_j is a query navigation operator; (2) $N_i = B(v_1, v_2)$ and $N_j = B(v_2, v_3)$; (3) $N_i = Q(q)$ and $N_j = B(Q(q), v)$.*

Definition 16 (Navigation Trace) *A navigation trace is a sequence of navigation operators $N_1, N_2, ..., N_k$ such that N_i and N_{i+1} are compatible.*

With these definitions, we can describe any user's information seeking process as a navigation trace. For example, if the user submits a query q_1, navigates into a region T_1 from the search result region, navigates further from T_1 to T_2, and finally submits another query q_2, then the process can be formally described by the navigation trace $Q(q_1), B(Q(q_1), T_1), B(T_1, T_2), Q(q_2)$. The flexibility of combining multiple operators formally to describe an arbitrary information seeking process shows the expressiveness of our framework. Indeed, it provides a solid theoretical basis for studying many different ways to combine querying and browsing as well as developing systems to integrate querying and browsing.

Viewing existing search engines in our navigation framework, we see that they mostly only support query navigation operators. A main contribution of our work is to study how to effectively support browsing navigation operators by a good topic map.

Ranking in the navigation framework. While not explored in this chapter, ranking is another important component in our framework. It is thus worth some discussion.

Ranking is important for three reasons. First, the ranking function is critical for supporting the query navigation operator as we generally define the target topic region of a query navigation operator as the top-ranked documents using the query. Second, even when a user reaches a region through browsing, it is still desirable to rank the documents in the region. As the user navigates from document to document within a region, the order of unseen documents can also be dynamically ordered as in the case of implicit feedback [61]. Third, when a user is landing on a region that is not exactly a region on our map, we will need to leverage the ranking function to find the closest regions on the map. A similar need also arises when the user takes a zoom-in or zoom-out action to change the resolution of the map, in which case a user may end up having multiple regions to choose.

While ranking of documents has been the central research topic in information retrieval and Web search, the navigation framework raises some new interesting research questions related to ranking: (1) Ranking documents within a topic region. In our framework, a user would leave a richer interaction history which would include not only queries, clickthroughs, but also browsing actions such as zoom in/out operations and neighborhood explorations. Existing work in personalized search and implicit feedback has already shown the usefulness of the existing query-based history information [61]. It would be very interesting to study how we can incorporate all the navigation information to further improve a ranking function. (2) Ranking topic regions. While traditionally, ranking is mainly to order documents, in the navigation framework, we also need to rank the topic regions of a map. How to generalize the current document ranking functions or design new ranking functions to perform region ranking is another very interesting research question. Some recent work on blog feeds has shown the promise of this research direction [22].

As a first step in studying the navigation framework, in this chapter, we simply reused the ranking function provided by an existing search engine, leaving all these questions for future work.

6.7 Conclusions

In this chapter, we study how to support flexible browsing for exploratory search. We define a novel multi-resolution topic map to extend a hierarchy to support more flexible browsing. We propose a novel way of exploiting and organizing search logs to enable users to follow information footprints left by other users in the process of information seeking, which can potentially lead to an interesting sustainable model for collaborative surfing. We also propose a general computational method based on the star-clustering algorithm to generate a multi-resolution topic map based on search logs. Experimental results using a sample of search logs from a commercial search engine show that browsing through such a search-log-based topic map is effective for supporting exploratory search.

Our work opens up many interesting new research directions as we have already discussed in Section 6.6. We think that it would be especially interesting to use a much larger data set of search logs to build a larger-scale topic map and evaluate its effectiveness with a system with real user traffic. It is also very interesting to study how to learn effectively from the rich interaction traces that a user leaves when interacting with a system that supports browsing with a topic map. Clearly more effective implicit feedback techniques can be developed by leveraging such interaction traces. Finally, topic maps can be constructed in multiple ways. How to design effective evaluation framework to compare different topic-map construction methods is an important research question.

Chapter 7

Support Effective Browsing II – Topic Maps on Plain Text Collections

7.1 Introduction

In the previous chapter, we introduced multi-resolution topic maps and proposed methods to construct such maps based on search logs. In this chapter, we study how to construct a topic map based on a plain text collection using a keyword-driven approach. To combine with querying, we propose novel methods to solve two challenges: (1) how to rank topic regions given a query and (2) how to rank documents inside a topic region in the context of querying. To test the effectiveness, we build topic maps on two TREC data sets and design experiments to simulate user interactions with a map. The simulation experiments show that our map can potentially significantly improve retrieval accuracy with very little user effort. More importantly, our experiments show that our topic maps can effectively help difficult queries, especially those mis-specified and under-specified queries.

7.2 Topic Map Construction

We formally defined multi-resolution topic maps in the previous chapter. In practice, a topic map can be constructed based on different data sources. The most promising and challenging data source is a plain text collection, which consists only of a set of documents. Such type of text collections is very common and usually very large. Searching information over this document collection is always difficult since it does not have any structure for a user to browse. Building a topic map over such a plain text collection is challenging but desirable.

7.2.1 Desired Properties

A topic map is to capture the global information structure of a text collection and to support effective browsing. There can be multiple ways of constructing such type of topic maps. In this section, we enumerate several desired properties of a topic map with the goal of supporting browsing.

- **Property 1**: Node label predictiveness. Each node in the map should have a content-bearing and meaningful label which clearly indicates the documents within the corresponding topic region.
- **Property 2**: Document coverage. Each level of the topic map should have a comprehensive coverage of the whole document collection.
- **Property 3**: Containment relationship. The hierarchy in a multi-resolution topic map should have clearly containment relationship and it is easy for users to understand the hierarchy structure.
- **Property 4**: Reachability. A better topic map can support a user to reach needed information flexibly. For example, there should be multiple possible paths to reach information in other places.

There are several possibilities to construct a topic map over a document collection. The most intuitive one is to use a document clustering algorithm to organize documents into a hierarchy and build a topic map based on the constructed hierarchy. However, such a method has a notable deficiency: It is notably hard to give a meaningful label to a cluster of documents since most of document clustering algorithms are *polythetic* and group documents into a cluster based on *multiple* keywords. A succinct cluster label can be un-informative or represents only a part of the documents in the cluster. In a topic map which has hierarchy structure, it is even harder to give discriminative labels for nodes in different levels. This is not desirable as for Property 1.

7.2.2 A Boolean Keyword Expression Approach

Considering all the desired properties, we propose a keyword based approach to build a topic map. Some previous work such as [57] has proposed to build a keyword hierarchy to organize search results. They determined the containment/subsumption relationship between two keywords based on their co-occurrence relationship. However, such an approach does not satisfy the Property 3. A keyword can be determined to subsume another keyword. But the topic region defined by the two keywords does not have a clear containment relationship. In the remaining of this section, we describe our way of constructing a topic map.

Map Node Definition

To capture the global information structure, we first need to identify those content-bearing keywords which spread over the whole collection. Nouns are always informative and we use nouns and frequent noun patterns as our map node candidates. Specifically, we define three types of nodes as follows:

- **Single-keyword node**: A single-keyword node contains only a single noun. For example, "flight" is a node. Such a node defines a topic region consisting of all the documents which contain "flight."
- **Multi-keyword AND node**: A multi-keyword AND node is a frequent noun pattern which consists of several nouns with a logic AND relation. We use the sign "+" to connect the keywords in a node. For example, "flight+passengers" is a node in this type. It defines a topic region consisting of documents which contain both "flight" AND "passengers."
- **Multi-keyword OR node**: A multi-keyword OR node consists of several keywords with a logic OR relation. We use the sign "|" to connect keywords in a node. For example, "flight|plane" is a node in this type. It defines a topic region consisting of documents which contain either "flight" OR "plane."

We can see all these node definitions can better satisfy Property 1: node label predictiveness.

Identify Map Nodes

Given the above definitions, all the nouns and their combination can be potentially map nodes. However, some combination of the keywords such as "flight+population" is not meaningful and may correspond to an empty topic region. Here, we describe how to identify meaningful map nodes. Given a keyword w, we use $D(w)$ as the set of document which contains word w and $|D(w)|$ as the size of $D(w)$.

Single-keyword node. All the noun words can be used as such a node. To be discriminative, such a word should not be too popular (e.g., "date" in news articles) or too rare. We use two parameters $maxDF$ and $minDF$ to filter out those undesired nouns. That is, we only keep nouns which satisfy

$$N_1 = \{w : w \text{ is a noun}, minDF \leq |D(w)| \leq maxDF\}.$$

Multi-keyword AND node. We use frequent pattern mining approach to identify such type of nodes [26]. Given a pattern p whose length is k, we generate a pattern of length $k+1$ as follows:

1. Identify the set of documents which contain p from the document collection C. We denote it as D_p.

2. For each $w \in N_1$, we compute the support and confidence of a pattern $p + w$ as $Supp = |D_p \bigcap D(w)|$ and $Conf = \frac{|D_p \bigcap D(w)|}{|D_p|}$.

3. If $Supp \geq minSupp$ and $Conf \geq minConf$, we keep the pattern $p + w$.

We iterate all the k-length patterns and generate a set of $(k+1)$-length patterns. Especially, when p is a single keyword pattern, i.e., $p \in N_1$, we can generate length-2 patterns following the above procedure. The two parameters $minSupp$ and $minConf$ are thresholds used to only keep those meaningful patterns.

Multi-keyword OR node. We identify those OR nodes using an algorithm which are similar to keyword clustering, based on a modified star clustering algorithm [3].

1. Order all the terms in N_1 in a decreasing order based on their document frequencies and get a ranked list R.

2. Sequentially select each *unmarked* element w in R and compute all its nearest neighbors of w by
$$sim(w, x) = \frac{|D(w) \bigcap D(x)|}{|D(w) \bigcup D(x)|}, \forall x \in R.$$

3. w and all x with $sim(w, x) \geq \sigma$ together make up an OR node. *Mark* all the keywords in this node.

4. Iterate 2 and 3 until there are no elements in R.

We have a parameter σ which is to control the coherence of a cluster of keywords. Suppose we get a set of OR nodes. The above procedure can be used recursively to produce another layer of OR nodes based on the current OR nodes.

Vertical and Horizontal Links

After we have a set of map nodes, we build a topic map by adding vertical and horizontal links among them.

It is clear that the definitions of our map nodes give clear hierarchy structure: Multi-keyword OR nodes subsume single-keyword nodes and single-keyword node contain multi-keyword AND nodes. For example, "flight|plane" is a parent of "flight," which is a parent of "flight+passengers." The topic regions defined by these nodes have clear containment relationship, which satisfies Property 3.

We build the horizontal links as follows. Given a node v, we use $D(v)$ to define its topic region, i.e., the set of documents which contain the the node v. Given two nodes u and v in the same level, we compute their closeness based on their overlapping area

$$sim(u, v) = \frac{|D(u) \bigcap D(v)|}{|D(u) \bigcup D(v)|}.$$

If $sim(u, v) \geq \delta$, we add a horizontal link between u and v.

Our proposed methods can satisfy all the desired properties listed before. Property 1 and Property 3 has been shown above. Our map also satisfies Property 2 and Property 4 well. Almost every document has nouns and our map nodes ensure a high document coverage. Our map contains both horizontal and vertical links and thus can reach a topic region through multiple noun sequences.

7.3 Integrating Topic Map with Querying

To support effective browsing based on our topic map in the context of queries, we need to address several problems in different stages of querying. The first stage is when a user submits a query, we need to display a list of map nodes for the user to start navigation on the map. When the user decides to select a node in the map, the map will be updated to show the surrounding nodes so that the user can continue browsing. When the user wants to view a topic region, how to rank the documents inside the current topic region in the context of query is another problem. We present our approaches in the following.

7.3.1 Ranking Map Nodes

Given a map node v in level $L(v)$, it defines a topic region consisting a set of documents $D(v)$. Given a query q, we rank node v by estimating the probability of reaching v from q. We derive it as follows:

$$P(v|q) = \sum_{d \in C} P(v|d,q)P(d|q) \approx \sum_{d \in D(v)} P(v|d)P(d|q).$$

The probability of $P(d|q)$ can be estimated by a standard language model approach. We estimate $P(v|d)$ as follows

$$P(v|d) = \frac{P(d,v)}{\sum_x P(d,x)} = \frac{1}{\sum_x I(d \in D(x))}$$

where $I(\cdot)$ is an indication function: $I(d \in D(x)) = 1$ if $d \in D(x)$ and 0 otherwise and $P(d,x) \propto I(d \in D(x))$.

For all the map nodes in the same level, we can rank them according to $P(v|q)$. Ideally, we need a way to decide which level of nodes to show automatically. In this chapter, we show the single-keyword nodes by default.

The above ranking function often biases toward big topic regions since it contains more documents. We define the bias of a topic region as $P(v) \propto |D(v)|$. We use the following method to overcome the bias. Suppose $P(v|q)$ is composed of two models: the true $\tilde{P}(v|q)$ and the bias $P(v)$. Given a interpolation parameter λ, we obtain $\tilde{P}(v|q)$ by

$$\min KLD\left(P(v|q)||(1-\lambda)\tilde{P}(v|q)+\lambda P(v)\right).$$

where $KLD(\cdot||\cdot)$ is KL-divergence function and λ is to control the bias and we set $\lambda = 0.9$ by default. Such an optimization problem can be solved very efficiently [91] and we omit the details here. We finally use $\tilde{P}(v|q)$ to rank the nodes.

7.3.2 Viewing a Map Node

When a user decides to view a map node, we need to display the map and show the results in the corresponding topic region with respect to the current query.

We use a fisheye view to display the map nodes. Setting the clicked node as the center, we display its horizontal neighbors, parents in a higher level and children in a lower level. The user can flexibly chooses to zoom in, zoom out, or move horizontally.

When a user reaches a topic region, our semantic of combining topic region with querying is to show the querying results confined in the current region. Formally, suppose we have a navigation trace starting from q: $\{q, T_1, ..., T_{k-1}\}$ where T_i are clicked map nodes after query q. When the user chooses a node T_k, we can rank the documents in the current region by

- Memoryless schema: For each $d \in T_k$, we use the original score of $score(d, q)$ as the score of d.
- Adaptive schema: In this schema, we always maintain a query model $\tilde{q}$ by incorporating all the viewed nodes of $T_1, ..., T_{k-1}$ and $score(d, \tilde{q})$use the score of d.

	#documents	#words	queries	#qrels
AP88-90	242918	306774	51-150	21829
ROBUST	528155	729384	310-450	14013

Table 7.1: Statistics of the two data sets.

We discuss an adaptive schema in language model approach. Since each T_i has a node which contains several keywords, we can have a bag of keywords by aggregating all the keywords together. Then a new query model $P(w|\tilde{q})$ can be computed as

$$P(w|\tilde{q}) = \alpha P(w|q) + (1-\alpha)P(w|T_1, ..., T_{k-1}).$$

where α is a parameter to balance the contribution from query and navigation trace.

7.4 Experiments

In this section, we evaluate the proposed topic map structure empirically based on several TREC data sets.

7.4.1 Experiment Design

We use two data sets: AP88-90 and ROBUST data sets. Table 7.1 show the statistics of these two data sets. For both of these data sets, we do not do any preprocessing such as stemming or stopword removal. For the AP data set, we use TREC ad-hoc topics 51–150 (a total of 100 topics) and for ROBUST data set, we use TREC ad-hoc topics 301–450 (a total of 150 topics).

To build the topic maps, we use a natural language processing package OpenNLP[1] to process each of these two data collections and obtain the Part-Of-Speech (POS) for all terms in the documents. After POS tagging, we keep only those nouns and analyze their co-occurrence relations to construct topic maps. There are several parameters when we build a topic map given a document collection and theses parameters are determined in an ad-hoc manner and fixed in the following experiments. That is, we do not modify the

[1]http://opennlp.sourceforge.net/

	minDF	maxDF	minSupp	minConf	σ
AP88-90	100	10000	100	0.05	0.1
ROBUST	400	20000	400	0.05	0.1

	OR-nodes		Single-nodes		AND-nodes		Total
	size	δ	size	δ	size	δ	–
AP88-90	932	0.2	9115	0.05	76842	0.2	86889
ROBUST	1677	0.1	10358	0.05	48760	0.1	60795

Table 7.2: The parameters used to build a topic map and the size of the map.

topic maps when conducting our simulation experiments. We build a three-level topic map for each collection. The three levels consist of a single-keyword, a multi-keyword AND, and a multi-keyword OR level. All the parameters and some statistics are summarized in Table 7.2.

It is challenging to test the effectiveness of a topic map. Following the methodology in [79], we evaluate the quality of our map based on simulation. The baseline method is the conventional language model approach with Dirichlet smoothing [90] (baseline). We set the smoothing parameter to 2000 in all the experiments as suggested in [90]. Another method is pseudo-feedback (pseudo-fb). We use the mixture method in the language modeling framework [89]. In pseudo-fb, we build a new query model based on the top 5 documents returned to the original query and use the interpolation parameter 0.9.

The power of browsing is that it can bring relevant documents on the top, instead of examining a long ranked list. Thus, to evaluate different approaches, we use Precision at 10 (P@10) measure as our main evaluation metric to judge the quality of top ranked results.

To evaluate our topic-map based methods, we design our simulation as follows. By default, we use the memoryless method as our way of ranking documents in a topic region. Given a query, we first provide 5 map nodes in single-keyword level and assume that the user will choose one of them. In this chapter, we conduct an upper-bound analysis and assume the user would choose the node whose P@10 of the ranked list inside the topic region is the best. We call this one-step-best browsing simulation (one-step-best). After the user views the first step node, we will display a portion of the map around the viewed node for the second step navigation and the user can choose the second node, forming a two-step browsing. We design several two-step browsing simulations to evaluate different types of

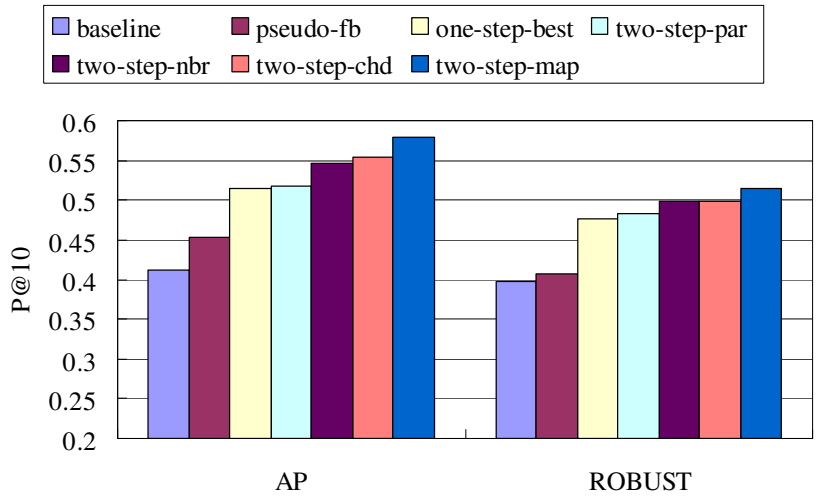

Figure 7.1: Overall comparison of different simulation methods.

links. The first simulation is to only allow the user to choose horizontal neighbors (two-step-nbr). The second is to only allow the user to choose hierarchical upper-level parents (two-step-par). The third is to only allow the user to choose lower-level children (two-step-chd). In our map, we have all these three types of links. We combine all these links together to evaluate the combined utility (two-step-map). For each of these simulations, we allow the user to examine up to 5 surrounding nodes for each type of links.

7.4.2 Experimental Results

We first give an overall comparison of the main methods we proposed in this chapter. Then we compare our map-based method with an optimal relevance term feedback method. Thirdly, we compare adaptive and memoryless schema when ranking documents inside a topic region. Finally, we analyze our methods with respect to query difficulty.

Overall Comparison

We compare the results of 7 methods based P@10 in Figure 7.1: baseline, pseudo-fb, one-step-best, two-step-par, two-step-nbr, two-step-chd, and two-step-map. In this figure, we show both the results on AP and ROBUST data sets. We have the following observations:

(1) Our map based methods can significantly outperform the baseline and pseudo-fb methods. For example, On AP and ROBUST respectively, we achieve relative improvements

Data set	#OR-node	#Single-node	#AND-node	Total
AP88-90	63	156	204	423
ROBUST	89	191	144	424

Table 7.3: The distribution of best nodes selected in two-step-map simulations.

over baseline 25.1% and 19.9% for one-step-best, and 40.7% and 29.7% for two-step-map. All these improvements are statistically significant based on Wilcoxin tests.

(2) Two-step explorations can significantly improve over one-step explorations. For example, on AP data set, two-step-map improves over one-step-best by 12.6% relatively (z-score=5.23); on ROBUST data set, two-step-map improves over one-step-best by 8.2% relatively (z-score=5.72). This shows that our topic maps can support users browse into better topic regions and further explorations are expected to bring more benefits to users.

(3) Comparing different types of links in two-step simulations, we can see that children and neighbors can achieve comparable results; both are better than parents. A possible reason is that parents are in relative larger topic regions and there are more distracting documents. Two-step-map combines all these links together and can outperform all these individual links. This shows that different types of links on our map are complementary. The horizontal links and vertical links provide flexible ways for users to reach different neighborhood.

Best node distribution. In Table 7.3, we show the distribution of the best selected nodes in our two-step-map simulations. Since there are ties among nodes, we count all the nodes which can achieve the best P@10. From this table, we can see that the best nodes are distributed in different levels. For example, on AP data set, 204 are from AND-nodes and 156 are from single-nodes. Such a diversity shows the usefulness and flexibility of our maps: multiple resolutions, horizontal links, and vertical links.

Comparison with Relevance Term Feedback

Our method allows a user to interactively select relevant terms from a map. In this sense, our method can be compared with a relevance term feedback approach since we used relevance judgments to simulate the traces of browsing. In this section, we compare our method with

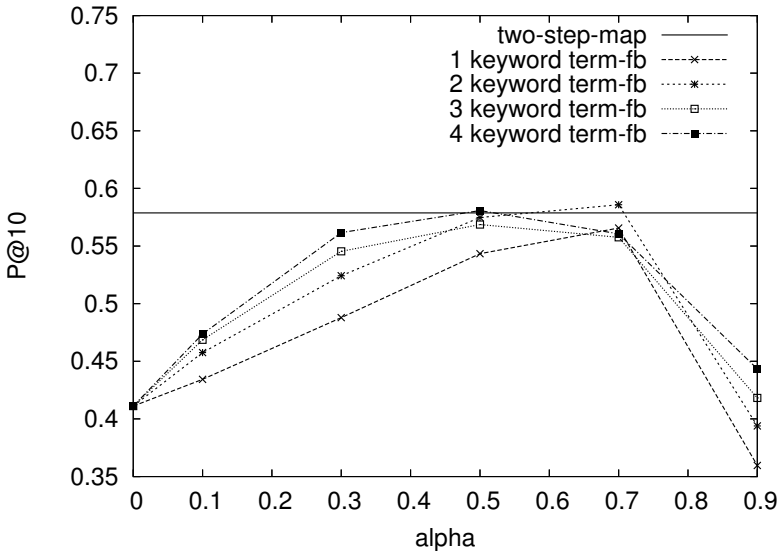

Figure 7.2: The performance of relevance term feedback.

an optimal relevance term feedback method (term-fb). For the term-fb method, given a query, we first extract the top 20 nouns from the top 20 initially ranked documents. We choose the 4 best nous from these top 20 as follows: For each of these 20 nouns, we append it to the original query and do a new retrieval. We evaluate the new ranked lists by P@10 and then select the best 4 nouns as feedback terms. Note that there is a very strong requirement in this method: Even in the relevance term feedback setting, a user can not easily select the top 4 keywords. Our goal here is to study how good our map-based method is, compared with this optimal term feedback setting.

After we get the feedback terms, we use the similar way to our adaptive schema to combine original query with them in the term-fb method. In Figure 7.2, we show the results on AP data by varying the number of feedback terms and the combination parameter α. From this figure, we can see that the best result obtained by relevance term feedback is comparable to ours. This means that our map-based method can achieve very similar results to this relevance term feedback. However, our map-based method only needs a user to make one or two clicks and does not need to tune any parameters. For the term-fb method, the figure shows that they are very sensitive to the combination parameter α. This shows another advantage of our map-based method in search refinement.

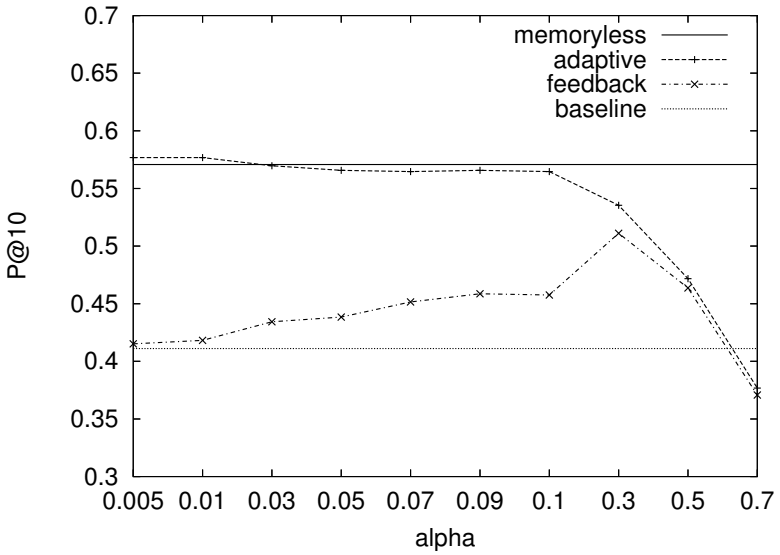

Figure 7.3: Adaptive method on AP data set.

Adaptive vs Memoryless

In this section, we compare the two schemas of ranking documents in a topic region: memoryless and adaptive. We also compare them with a feedback method by augmenting queries with the node labels. The difference between our adaptive method and the feedback method is that the feedback method ranks the documents in the whole collection but our adaptive method only ranks the documents in the topic region.

In Figure 7.3 and Figure 7.4, we vary the parameter α for both the adaptive method and the feedback method and show the results on both AP and ROBUST data sets. These two figures show that the adaptive method can hardly improve over the memoryless one. This means that using node labels is not very effective when reranking documents inside topic regions. This might be because we already use the node labels to define the topic regions and thus redundantly using such information for reranking can not help much. Compared with the memoryless or adaptive method, the feedback method works very poor. This is because node labels only consist of 1-2 keywords and the feedback method can be easily biased towards these expanded terms. Our method provides a natural search refinement mechanism through effective browsing.

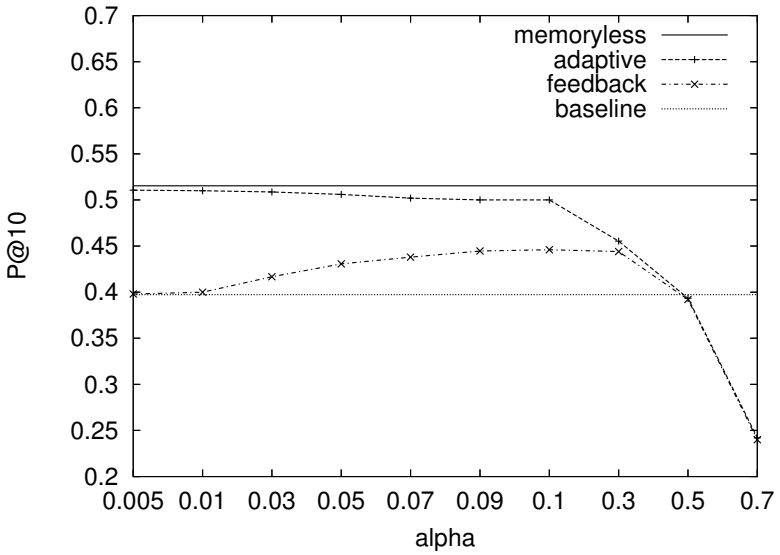

Figure 7.4: Adaptive method on ROBUST data set.

Query Difficulty

In this section, we compare methods with respect to query difficulty. We use the P@10 values of baseline method to quantify query difficulty and separate queries into 11 bins. All the queries in each bin have the same P@10 in our baseline method (ranging from 0 to 1 with step 0.1).

Figure 7.5 and Figure 7.6 show the results of baseline, pseudo-feedback, relevance term feedback, and our map-based method. These two figures have very similar trends. We can see that our method and the term feedback method consistently improve over both the baseline and the pseudo-feedback method. It is very encouraging to observe that our method can improve those most difficult queries from P@10=0 to P@10=0.13 on AP and from P@10=0 to P@10=0.15 on ROBUST data set. These improvements are larger than the term feedback method for the most difficult queries. This is much desirable since for difficult queries, supporting browsing is more promising than soliciting feedback. Our topic maps are very effective in supporting browsing and can significantly improve retrieval accuracy, with very little user effort.

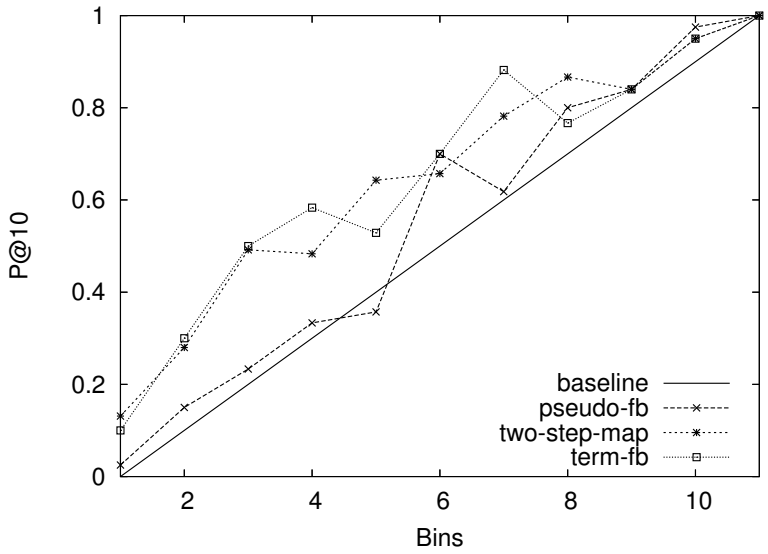

Figure 7.5: Performance with respect to query difficulty on AP data.

7.4.3 Case Study

In this section, we show several examples including a portion of our constructed map and the navigation traces of several queries.

A Portion of the Map

In this section, we show a portion of the map we constructed on the AP data set. Figure 7.7 shows the map around "flight." From this figure, we can see that different levels correspond to topic regions in different resolutions. In the same level, we have horizontal neighbors. The neighbors in a higher level have larger distances, while the neighbors in a lower level are pretty close to each other. They define overlapping but different topic regions .

This example shows that such a keyword map not only has a clear notion of topic regions, but also clear notion of levels. On the other hands, such a map provides terms which is complementary to the conventional search engines which provide documents. The users can flexibly interact with documents or map to refine their search naturally.

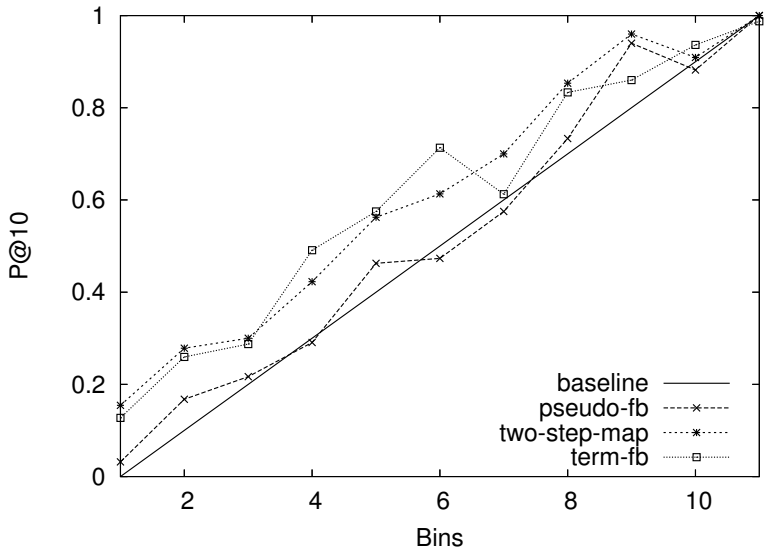

Figure 7.6: Performance with respect to query difficulty on ROBUST data.

Topic ID	query keywords	P@10
67	politically motivated civil disturbances	0
74	conflicting policy	0
314	marine vegetation	0
326	ferry sinkings	0
436	railway accidents	0

Table 7.4: Query examples.

Navigation Trace Examples

Roughly speaking, difficult queries can be classified as over-specified, under-specified, or mis-specified. We found that most of queries in TREC are under-specified or mis-specified. In Table 7.4 and Table 7.5, we show several examples of difficult queries and how our maps can help improve their result accuracy. From these two tables, we can see that our topic map can help refine a vague query or address the vocabulary mismatch problem. For example, topic 74 is very vague. Our map can suggest to visit node of "license" and thus make the query more specific. Topic 314 is difficult due to vocabulary mismatch and our topic map can help address this by providing "seas" or "oceans" to deemphasize on the mismatched word "marine." All these examples show the promise of our approach to improve the most difficult topics.

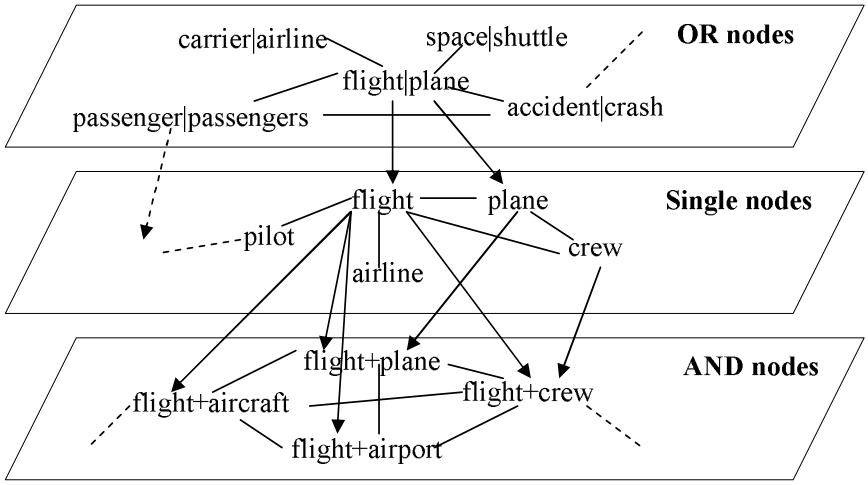

Figure 7.7: A portion of the topic map based on AP data set.

Topic ID	1st node	P@10	2nd node	P@10
67	uprising	0.1	uprising+clashes	0.3
74	license	0.1	license|plates	0.1
314	seas	0.1	oceans	0.2
326	sea	0.1	sea+fleet	0.3
436	train	0.3	train+station	0.5

Table 7.5: Navigation trace examples.

7.5 Conclusions and Future Work

Browsing is an complimentary information access mechanism to direct querying and has become more important for difficult queries. In this chapter, we studied how to support effective browsing during a search process. We proposed a novel multi-resolution topic map structure and a practical way to construct such a map from a simple text collection. We further proposed method to integrate our topic maps with querying. Experiments with two TREC data sets show that the proposed topic maps are very promising. With very little user effort, it can improve the retrieval accuracy significantly.

Our current work can be extended in several interesting directions. First, our current simulation based experiments show the promise of our topic maps. We plan to apply our techniques to build prototype systems and further evaluate them based on real user statistics. Second, we can explore other ways of constructing topic maps and compare them. Third, when a user leaves rich interaction traces with search results, especially

click-through information, how to leverage this interaction traces to design better adaptive ranking is an interesting problem.

Chapter 8

Conclusions and Outlook

This dissertation studied how to improve Web search for difficult queries. Due to the lack of necessary domain knowledge or complex information needs, a user can often encounter difficulty to compose effective queries. As a result, the composed queries are ineffective with the following reasons: ambiguity, vocabulary mismatch, and lack of discrimination. I proposed to address these problems systematically from multiple perspectives to help users find hard-to-find information. Specifically, I proposed to improve search for difficult queries by: effective query reformulation, user-oriented search result organization, actively negative relevance feedback, and effective browsing support. All the methods are novel, effective, and efficient. Experimental results showed that our proposed methods can significantly improve search quality for difficult queries.

This dissertation is the first systematic study of difficult queries. It opens up many interesting research directions for future study:

Adaptive support for difficult queries. The study of difficult queries has just attracted a lot of attention recently. In my dissertation, I have developed techniques that are shown to be effective for difficult queries, but they sometimes are ineffective for easy queries. Besides, depending on the root causes of the difficulty, we may need to use different techniques. It would thus be interesting to study how to automatically categorize queries based on root causes and automatically adapt/customize the strategy to improve search results for difficult queries based on identified causes. The current understanding of the root causes of difficult queries are still limited in a coarse granularity. In the future, constructing a comprehensive and finer-granularity taxonomy of difficult queries would have high impact. Based on the taxonomy, machine learning techniques can be leveraged to identify or categorize difficult queries into the taxonomy. For a particular category in the taxonomy, we

can further study specific retrieval models, presentation strategies, and feedback techniques to improve search utilities for a family of difficult queries.

Rich user interactions. User interactions can provide valuable information to help difficult queries. Our topic maps provide more functionalities for users to interact with our system. As a result, the user would leave rich exploration traces. How to further enhance users' interactions and leverage users' traces is an interesting problem. For example, with enough traces, data mining techniques (e.g., frequent sequence mining) can be used to recommend meaningful traces for future users. Traces of users in a social network can help their adjacent friends' information seeking processes.

Unified search paradigm. In my dissertation, I studied different approaches to address difficult queries from different perspectives. Their effectiveness is shown individually. A natural question is how to integrate all these approaches together in a single system. For example: How to seamlessly integrate querying and browsing together? How to show query reformulation recommendations, organized results, and topic maps in a unified interface? How to selectively choose different approaches for difficult queries with different causes? How to educate users to understand and use such an integrate system naturally? These directions involve interesting research questions such as layout optimization, human-computer interface (HCI) design, decision making, redundancy control, etc. Such a system makes it possible to collect massive user feedback and user traces, which enable us to build a "social surfing" system where past user interactions can benefit future information seekers.

In summary, this dissertation laid down a framework to study difficult queries. This can inspire many interesting future research directions and facilitate future study of how to further improve overall search quality. We anticipate that a more intelligent system can integrate all the approaches together and eventually change information access paradigm.

References

[1] E. Agichtein, E. Brill, and S. T. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, pages 19–26, 2006.

[2] P. Anick. Using terminological feedback for web search refinement: a log-based study. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 88–95, 2003.

[3] J. A. Aslam, E. Pelekov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *Journal of Graph Algorithms and Applicatins*, 8(1):95–129, 2004.

[4] R. Attar and A. S. Fraenkel. Local feedback in full-text retrieval systems. *J. ACM*, 24(3):397–417, 1977.

[5] R. A. Baeza-Yates. Applications of web query mining. In *ECIR*, pages 7–22, 2005.

[6] M. J. Bates. The design of browsing and berrypicking techniques for the online search interface. *Online Review*, 13:407–424, 1989.

[7] D. Beeferman and A. L. Berger. Agglomerative clustering of a search engine query log. In *KDD*, pages 407–416, 2000.

[8] N. J. Belkin. Interaction with texts: Information retrieval as information-seeking behavior. In *Information Retrieval*, pages 55–66, 1993.

[9] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[10] C. Buckley. Why current ir engines fail. In *SIGIR'04*, pages 584–585, 2004.

[11] D. Carmel, E. Yom-Tov, A. Darlow, and D. Pelleg. What makes a query difficult? In *SIGIR*, pages 390–397, 2006.

[12] H. Chen and S. T. Dumais. Bringing order to the web: automatically categorizing search results. In *CHI*, pages 145–152, 2000.

[13] S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 2–11, 2005.

[14] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proceedings of ACM SIGIR 2002*, pages 299–306, 2002.

[15] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *WWW*, pages 325–332, 2002.

[16] E. Cutrell, D. Robbins, S. Dumais, and R. Sarin. Fast, flexible filtering with phlat. In *CHI*, pages 261–270, 2006.

[17] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *SIGIR*, pages 318–329, 1992.

[18] W. Dakka and P. Ipeirotis. Automatic extraction of useful facet terms from text documents. In *ICDE*, 2008.

[19] S. T. Dumais, E. Cutrell, and H. Chen. Optimizing search by showing results in context. In *CHI*, pages 277–284, 2001.

[20] M. Dunlop. The effect of accessing non-matching documents on relevance feedback. *ACM TOIS*, 15(2), 1997.

[21] E. Duval and H. Olivié. Towards the integration of a query mechanism and navigation for retrieval of data on multimedia documents. *SIGIR Forum*, 26(2):8–25, 1992.

[22] J. L. Elsas, J. Arguello, J. Callan, and J. G. Carbonell. Retrieval and feedback models for blog feed search. In *SIGIR*, pages 347–354, 2008.

[23] J. English, M. A. Hearst, R. R. Sinha, K. Swearingen, and K.-P. Yee. Hierarchical faceted metadata in site search interfaces. In *CHI Extended Abstracts*, pages 628–639, 2002.

[24] D. A. Evans and R. G. Lefferts. Design and evaluation of the CLARIT TREC-2 system. In D. Harman, editor, *TREC-2*, pages 137–150, 1994.

[25] G. W. Furnas. Effective view navigation. In *CHI*, pages 367–374, 1997.

[26] J. Han and M. Kamber. *Data Mining: Concepts and Techniques, 2nd Ed.* Morgan Kaufmann, 2006.

[27] D. Harman and C. Buckley. SIGIR 2004 Workshop: RIA and where can IR go from here. *SIGIR Forum*, 38(2):45–49, 2004.

[28] M. A. Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4):59–61, 2006.

[29] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *SIGIR*, pages 76–84, 1996.

[30] D. Inkpen and G. Hirst. Building and using a lexical knowledge-base of near-synonym differences. *Computational Linguistics*, 32(2):223–262, June 2006.

[31] M. Iwayama. Relevance feedback with a small number of relevance judgements: incremental relevance feedback vs. document clustering. In *SIGIR*, pages 10–16, 2000.

[32] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[33] T. Joachims. *Evaluating Retrieval Performance Using Clickthrough Data.*, pages 79–96. Physica/Springer Verlag, 2003. in J. Franke and G. Nakhaeizadeh and I. Renz, "Text Mining".

[34] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.

[35] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. On semi-automated web taxonomy construction. In *WebDB*, 2001.

[36] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW*, pages 658–665, 2004.

[37] J. D. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR*, pages 111–119, 2001.

[38] m. c. schraefel, D. A. Smith, A. Owens, A. Russell, C. Harris, and M. Wilson. The evolving mspace platform: leveraging the semantic web on the trail of the memex. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 174–183, 2005.

[39] Y. S. Maarek. Organizing documents to support browsing in digital libraries. *SIGOIS Bull.*, 16(2):36–37, 1995.

[40] J. D. Mackinlay and P. T. Zellweger. Browsing vs. search: can we find a synergy? (panel session). In *CHI*, pages 179–180, 1995.

[41] G. Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, 2006.

[42] Microsoft Live Labs. Accelerating search in academic research, 2006. http://research.microsoft.com/ur/us/fundingopps/RFPs/Search_2006_RFP.aspx.

[43] V. L. O'Day and R. Jeffries. Orienteering in an information landscape: how information seekers get from here to there. In *INTERCHI*, pages 438–445, 1993.

[44] C. Olston and E. H. Chi. Scenttrails: Integrating browsing and searching on the web. *ACM Trans. Comput.-Hum. Interact.*, 10(3):177–197, 2003.

[45] S. Pandit and C. Olston. Navigation-aided retrieval. In *WWW*, pages 391–400, 2007.

[46] F. Peng, N. Ahmed, X. Li, and Y. Lu. Context sensitive stemming for web search. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 639–646, 2007.

[47] P. Pirolli, P. K. Schank, M. A. Hearst, and C. Diehl. Scatter/gather browsing communicates the topic structure of a very large text collection. In *CHI*, pages 213–220, 1996.

[48] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *KDD*, pages 239–248, 2005.

[49] R. Rapp. The computation of word associations: comparing syntagmatic and paradigmatic approaches. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[50] S. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146, 1976.

[51] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR*, pages 232–241, 1994.

[52] S. E. Robertson, S. Walker, S. Jones, M. M.Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In D. K. Harman, editor, *The Third Text REtrieval Conference (TREC-3)*, pages 109–126, 1995.

[53] J. J. Rocchio. Relevance feedback in information retrieval. In *The SMART Retrieval SystemExperiments in Automatic Document Processing*, pages 313–323, 1971.

[54] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW*, pages 377–386, 2006.

[55] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *JASIS*, 44(4):288–297, 1990.

[56] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[57] M. Sanderson and B. Croft. Deriving concept hierarchies from text. In *SIGIR*, pages 206–213, 1999.

[58] J. Savoy. Why do successful search systems fail for some topics. In *SAC'07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 872–877, 2007.

[59] D. Shen, M. Qin, W. Chen, Q. Yang, and Z. Chen. Mining web query hierarchies from clickthrough data. In *AAAI*, pages 341–346, 2007.

[60] R. Shen, N. S. Vemuri, W. Fan, R. da S. Torres, and E. A. Fox. Exploring digital libraries: integrating browsing, searching, and visualization. In *JCDL '06: Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 1–10, 2006.

[61] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR*, pages 43–50, 2005.

[62] A. Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43, 2001.

[63] A. Singhal, M. Mitra, and C. Buckley. Learning rounting queries in a query zone. In *Proceedings of ACM SIGIR 1997*, pages 25–32, 1997.

[64] C. J. van Rijsbergen. *Information Retrieval, second edition.* Butterworths, London, 1979.

[65] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, Berlin, 1995.

[66] M. Vlachos, C. Meek, Z. Vagena, and D. Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *SIGMOD*, pages 131–142, 2004.

[67] E. M. Voorhees. Draft: Overview of the trec 2005 robust retrieval track. In *Notebook of TREC2005*, 2005.

[68] E. M. Voorhees. Overview of the trec 2004 robust retrieval track. In *TREC2004*, 2005.

[69] X. Wang, H. Fang, and C. Zhai. Improve retrieval accuracy for difficult queries using negative feedback. In *CIKM*, pages 991–994, 2007.

[70] X. Wang, H. Fang, and C. Zhai. A study of methods for negative relevance feedback. In *SIGIR*, pages 219–226, 2008.

[71] X. Wang, J.-T. Sun, Z. Chen, and C. Zhai. Latent semantic analysis for multiple-type interrelated data objects. In *SIGIR*, pages 236–243, 2006.

[72] X. Wang and C. Zhai. Learn from web search logs to organize search results. In *SIGIR*, pages 87–94, 2007.

[73] X. Wang and C. Zhai. Massive implicit feedback: Organizing search logs into topic maps for collaborative surfing. In *ACM SIGIR workshop on Understanding the Users (Demo Description)*, 2009.

[74] J.-R. Wen, J.-Y. Nie, and H. Zhang. Clustering user queries of a search engine. In *WWW*, pages 162–168, 2001.

[75] A. Wexelblat and P. Maes. Footprints: history-rich tools for information foraging. In *CHI*, pages 270–277, 1999.

[76] R. W. White, S. M. Drucker, G. Marchionini, M. Hearst, and m.c. schraefel. Exploratory search and HCI. In *Proceedings of SIGCHI 2007 Workshop*, 2007.

[77] R. W. White, B. Kules, and S. M. D. m.c. schraefel. Supporting exploratory search, introduction, special issue, communications of the ACM. *Commun. ACM*, 49(4):36–39, 2006.

[78] R. W. White and R. A. Roth. *Exploratory Search: Beyond the Query-Response Paradigm.* Morgan and Claypool, 2009.

[79] R. W. White, I. Ruthven, J. M. Jose, and C. J. V. Rijsbergen. Evaluating implicit feedback models using searcher simulations. *ACM Trans. Inf. Syst.*, 23(3):325–361, 2005.

[80] J. Xu and B. Croft. Query expansion using local and global document analysis. In *Proceedings of ACM SIGIR 1996*, pages 4–11, 1996.

[81] J. Xu and W. B. Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Trans. Inf. Syst.*, 18(1):79–112, 2000.

[82] Z. Xu and R. Akella. Active relevance feedback for difficult queries. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge mining*, pages 459–468, 2008.

[83] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI*, pages 401–408, 2003.

[84] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow. Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In *SIGIR*, pages 512–519, 2005.

[85] X. Yuan and N. J. Belkin. Supporting multiple information-seeking strategies in a single system framework. In *SIGIR*, pages 247–254, 2007.

[86] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *SIGIR*, pages 46–54, 1998.

[87] O. Zamir and O. Etzioni. Grouper: A dynamic clustering interface to web search results. *Computer Networks*, 31(11-16):1361–1374, 1999.

[88] H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, and J. Ma. Learning to cluster web search results. In *SIGIR*, pages 210–217, 2004.

[89] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of ACM CIKM 2001*, pages 403–410, 2001.

[90] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of ACM SIGIR 2001*, pages 334–342, 2001.

[91] Y. Zhang and W. Xu. Fast exact maximum likelihood estimation for mixture of language models. In *SIGIR*, pages 865–866, 2007.

Author's Biography

Xuanhui Wang was born in Yongji city, Shanxi province, People's Republic of China. He attended the University of Science and Technology of China (USTC), where he earned the bachelor of engineering in computer science in 2003. He received scholarship every year during his undergraduate study in USTC.

Dr. Wang subsequently entered the graduate program of the University of Illinois at Urbana-Champaign (UIUC). He obtained his master of science degree in computer science in 2006 with his thesis entitled *Latent Semantic Analysis for Multiple-Type Interrelated Data Objects*.

Dr. Wang further pursued his doctoral degree in the area of information retrieval at UIUC under the supervision of Dr. ChengXiang Zhai. He published around 20 papers and filed several patents. He was one of the 4 recipients of Yahoo! Ph.D. Fellowship during 2008-2009. He received his Ph.D. from UIUC in 2009 with his dissertation entitled *Improving Web Search for Difficult Queries*.